IBM Research

IBM High Performance Computing Toolkit

I-Hsin Chung

IBM T.J. Watson Research Center

ihchung@us.ibm.com

What is it?

- **IBM long-term goal:**
  - An automatic performance tuning framework
    - Assist users to identify performance problems
    - Provide possible solutions
  - A common application performance analysis environment across all HPC platforms
  - Look at all aspects of performance (communication, memory, processor, I/O, etc) from within a single interface
- **Where we are: one consolidated package**
  - One consolidate package (AIX, Linux/Power)
  - Tools for MPI, OMP, processor, memory etc
  - Operate on the binary and yet provide reports in terms of source-level symbols
  - Dynamically activate/deactivate data collection and change what information to collect
  - One common visualization GUI

IBM High Performance Computing Toolkit on Blue Gene

- **MPI performance: MPI Profiler/Tracer**
- **CPU performance: Xprofiler, HPM**
- **Threading performance: OpenMP profiling**
- **I/O performance: I/O profiling**
- **Visualization and analysis: PeekPerf**

Supported Platforms

- **AIX**
- **Linux**
  - PowerPC
  - Blue Gene /L and Blue Gene /P
  - Intel x86 & AMD (planned)
- **Eclipse integration**
- **Windows (Intel/AMD) + Mac (coming soon)**
  - Offline Peekperf visualization capability only

AGENDA

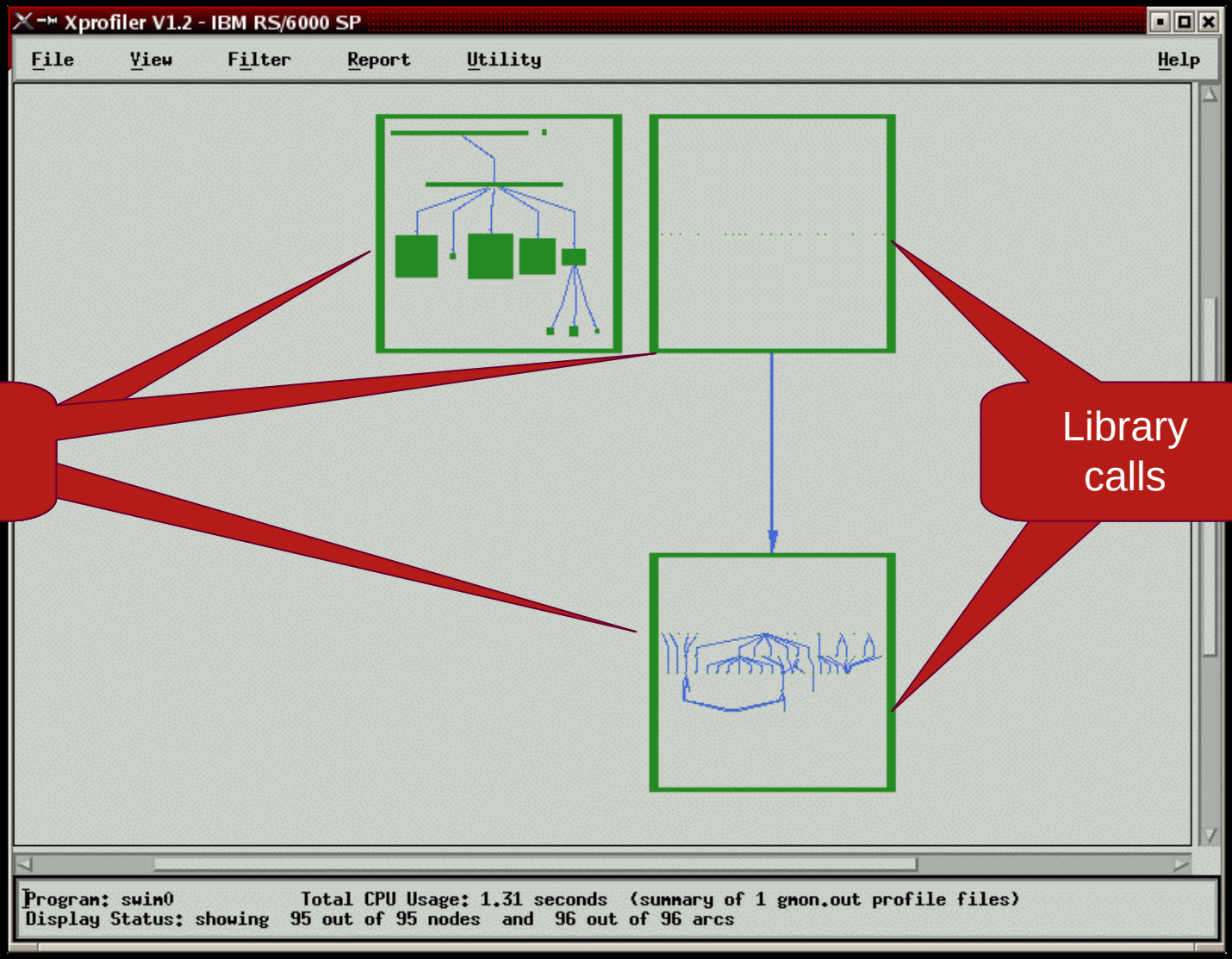
- **Xprofiler: call-graph profiling**
- **HPM: hardware counter data**
- **MPI Profiler/Tracer: MPI profiling**
- **PompProf: OpenMP profiling**
- **MIO: I/O profiling and optimization**
- **IBM HPC Toolkit**
- **Questions/Comments**

XProfiler

Xprofiler

- **Visualizer CPU time profiling data**
- **Compile and link with -g -pg flags + optimization**
- **Code execution generates gmon.out file**
  - MPI applications generate gmon.out.1, …, gmon.out.n
- **Analyze gmon.out file with Xprofiler**
  - xprofiler a.out gmon.out
- **Important factors:**
  - Sampling interval is in the order of ms
  - Profiling introduces overhead due to function calls

Xprofiler - Initial View

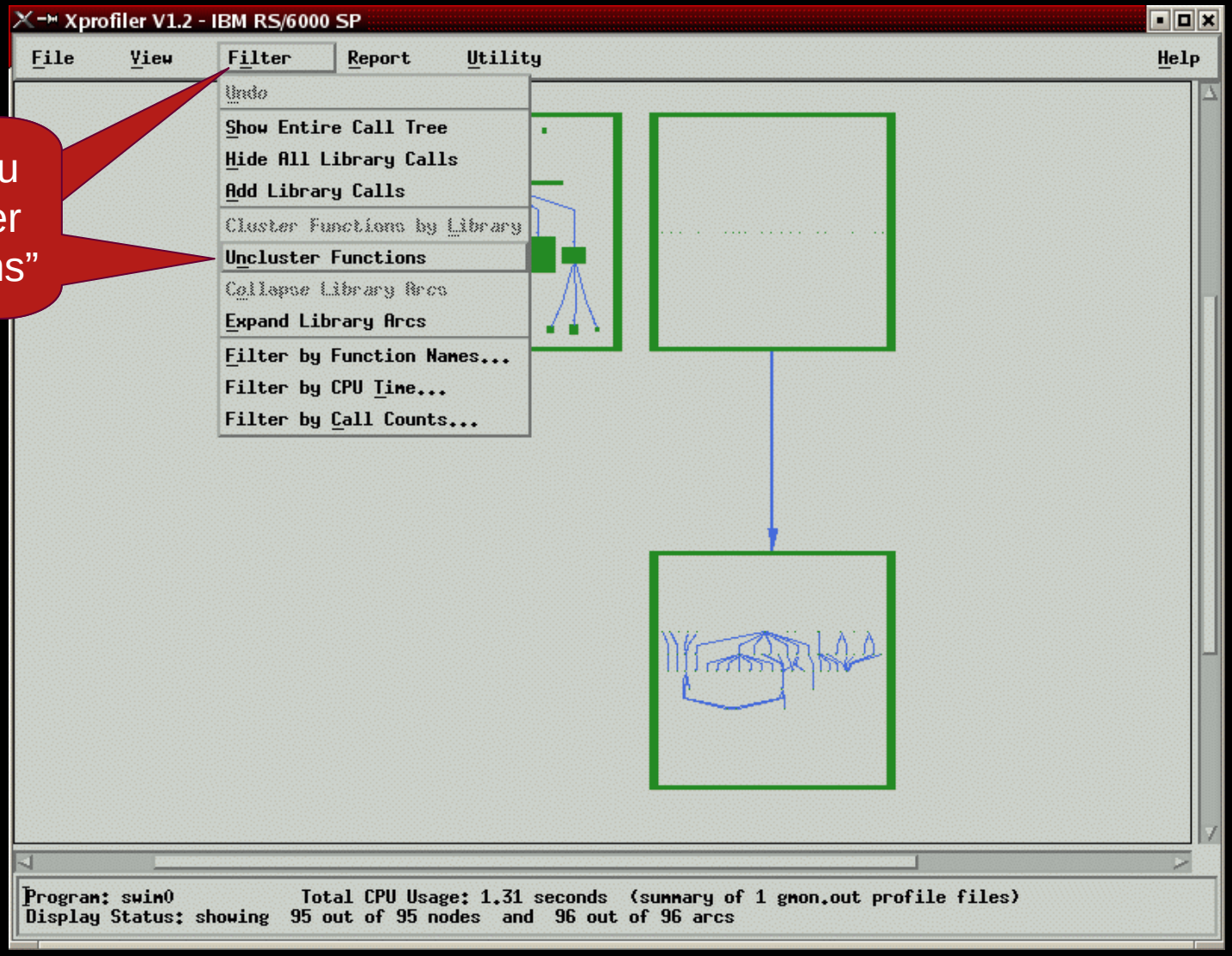


Clustered functions

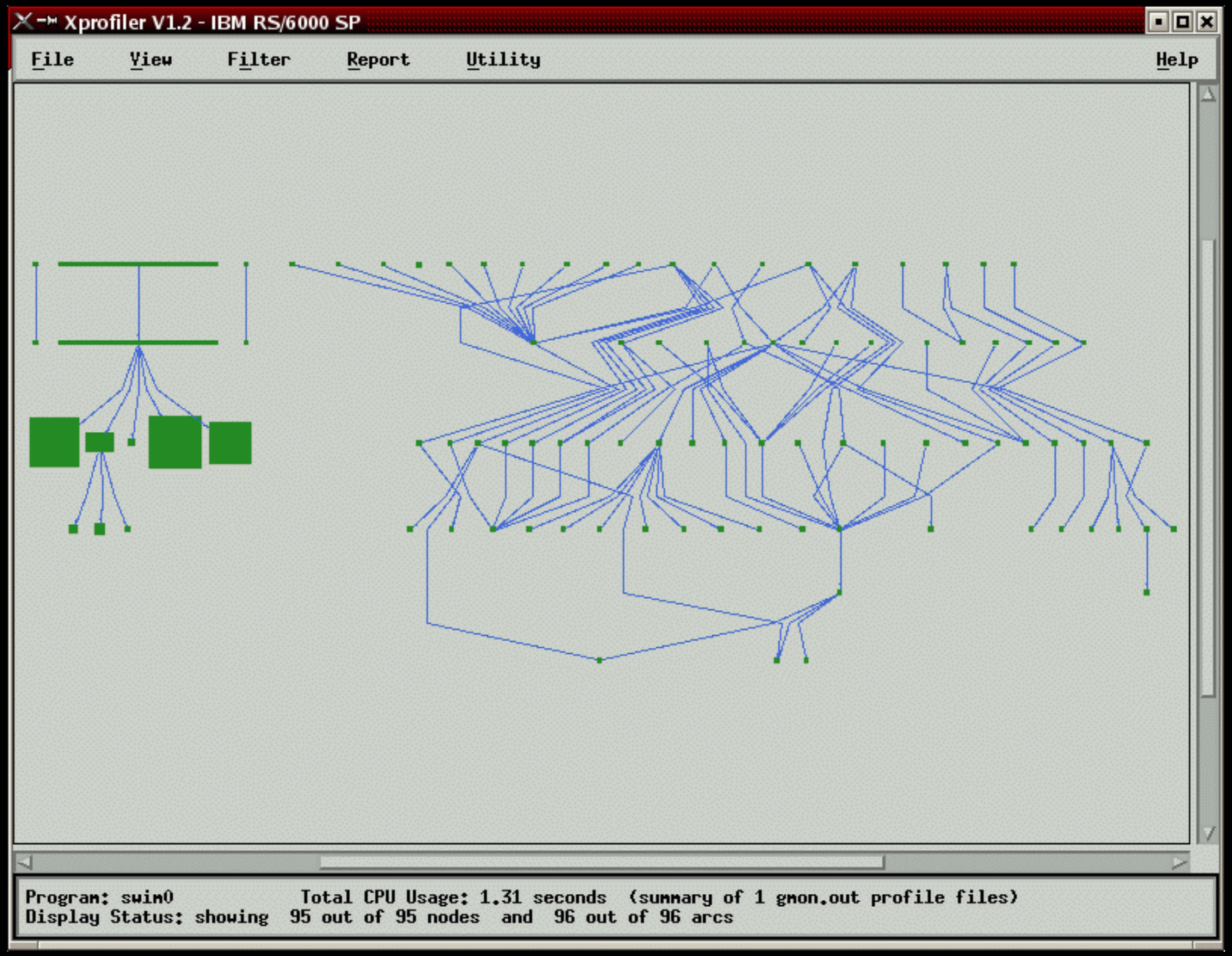
Library calls

Xprofiler - Unclustering Functions

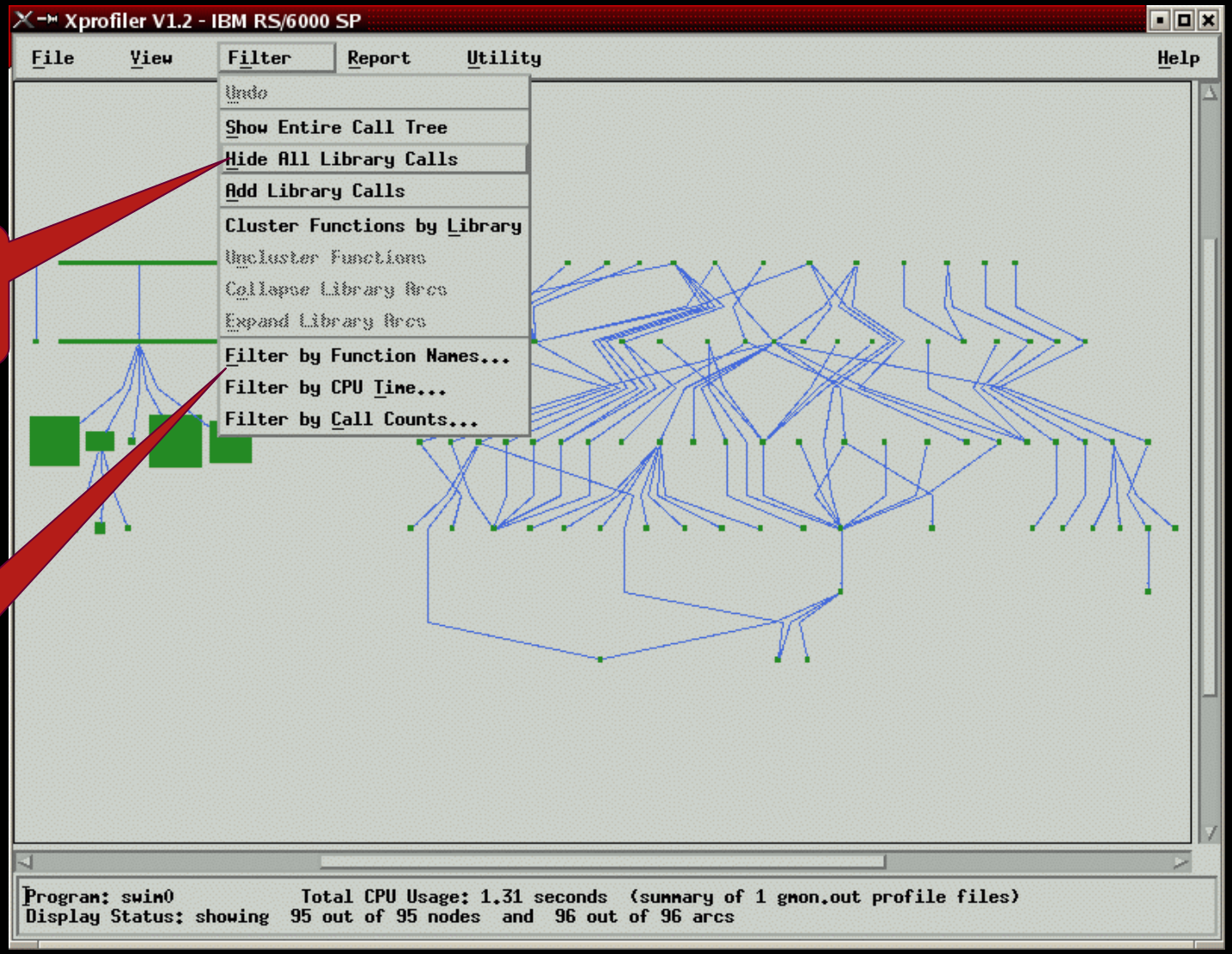
on “Filter” menu select “Uncluster Functions”



Xprofiler - Full View - Application and Library Calls



Xprofiler - Hide Lib Calls Menu

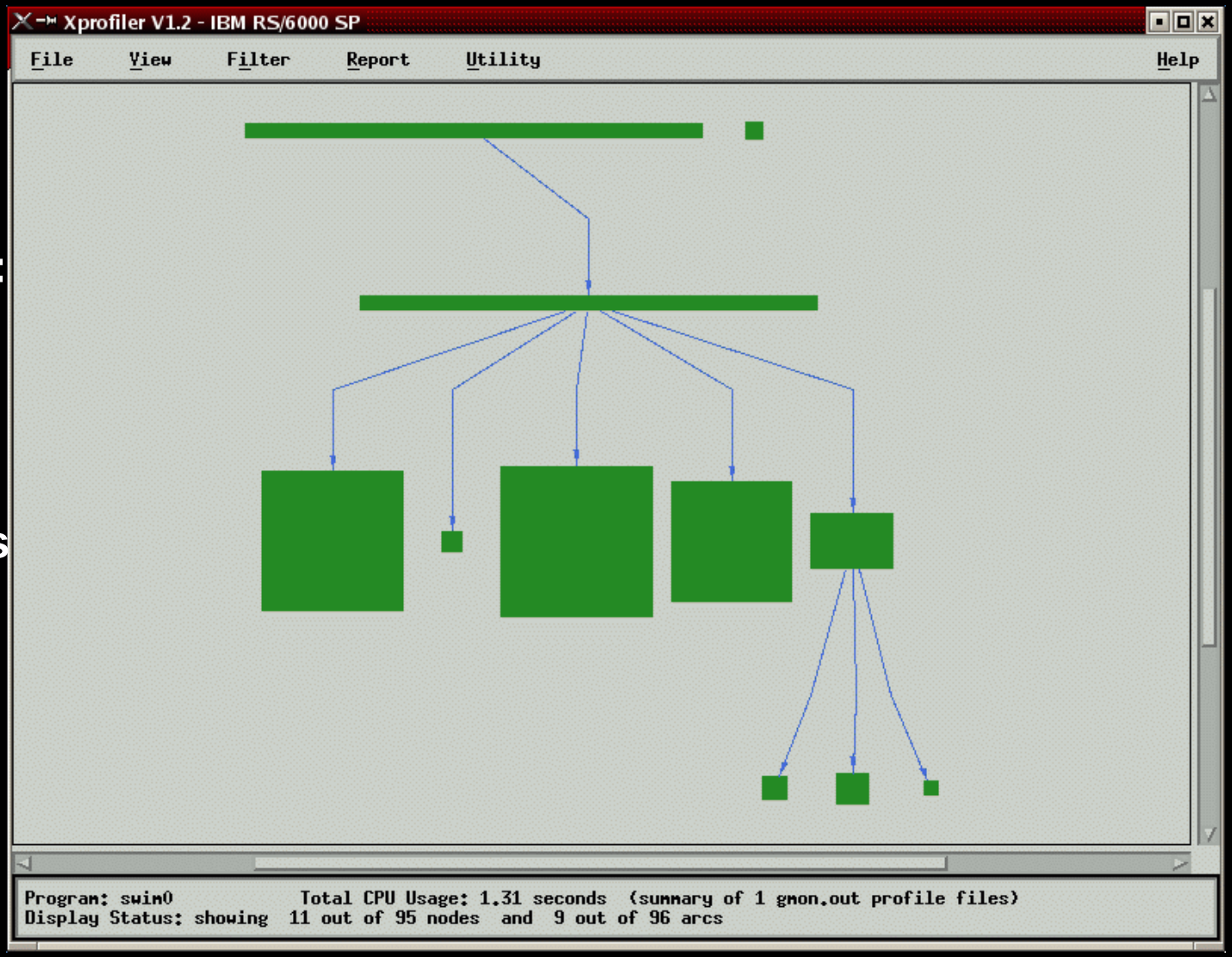


Now select "Hide All Library Calls"

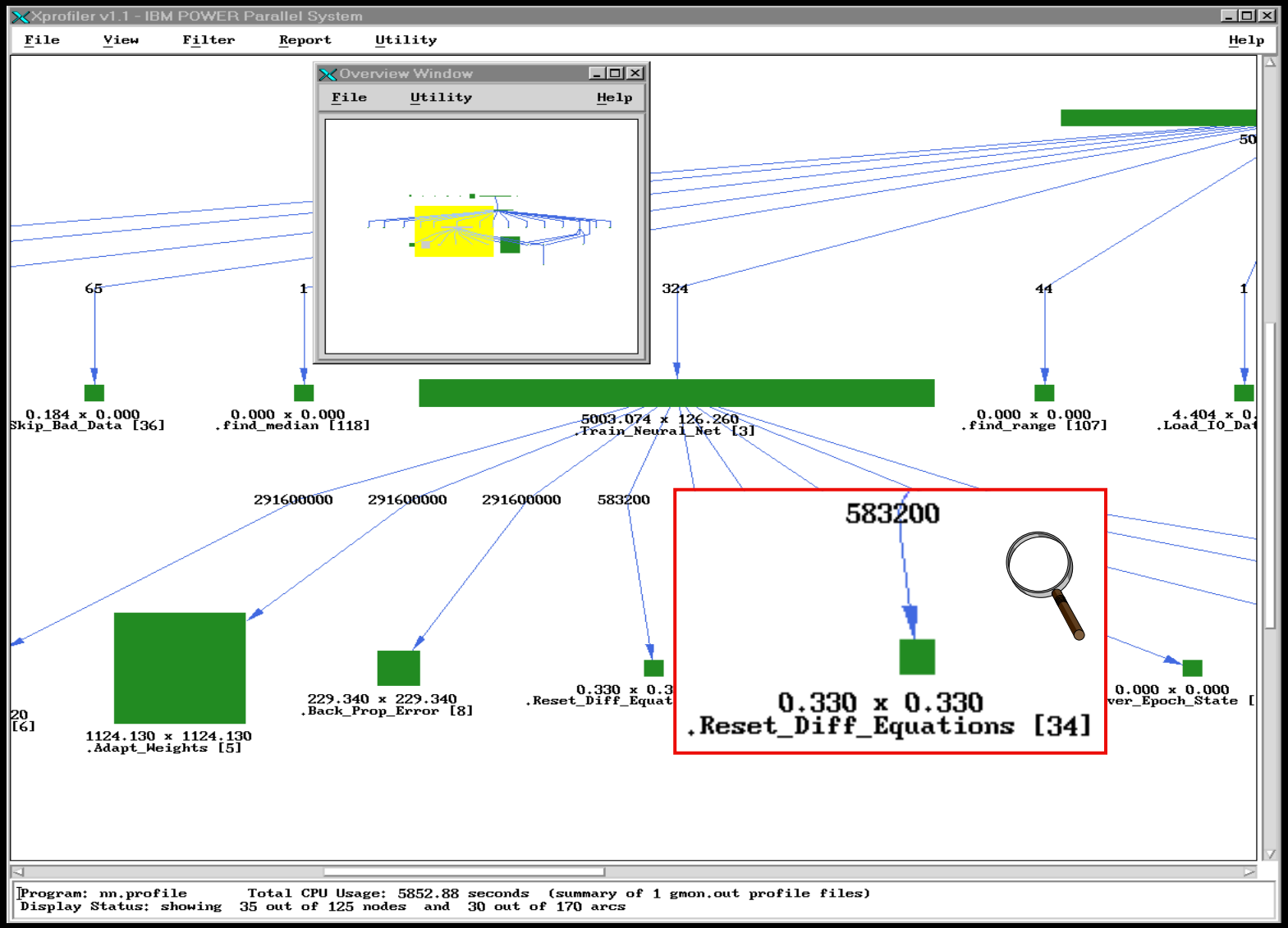
Can also filter by:
Function Names,
CPU Time,
Call Counts

Xprofiler - Application View

- **Width of a bar: time including called routines**
- **Height of a bar: time excluding called routines**
- **Call arrows labeled with number of calls**
- **Overview window for easy navigation (View → Overview)**

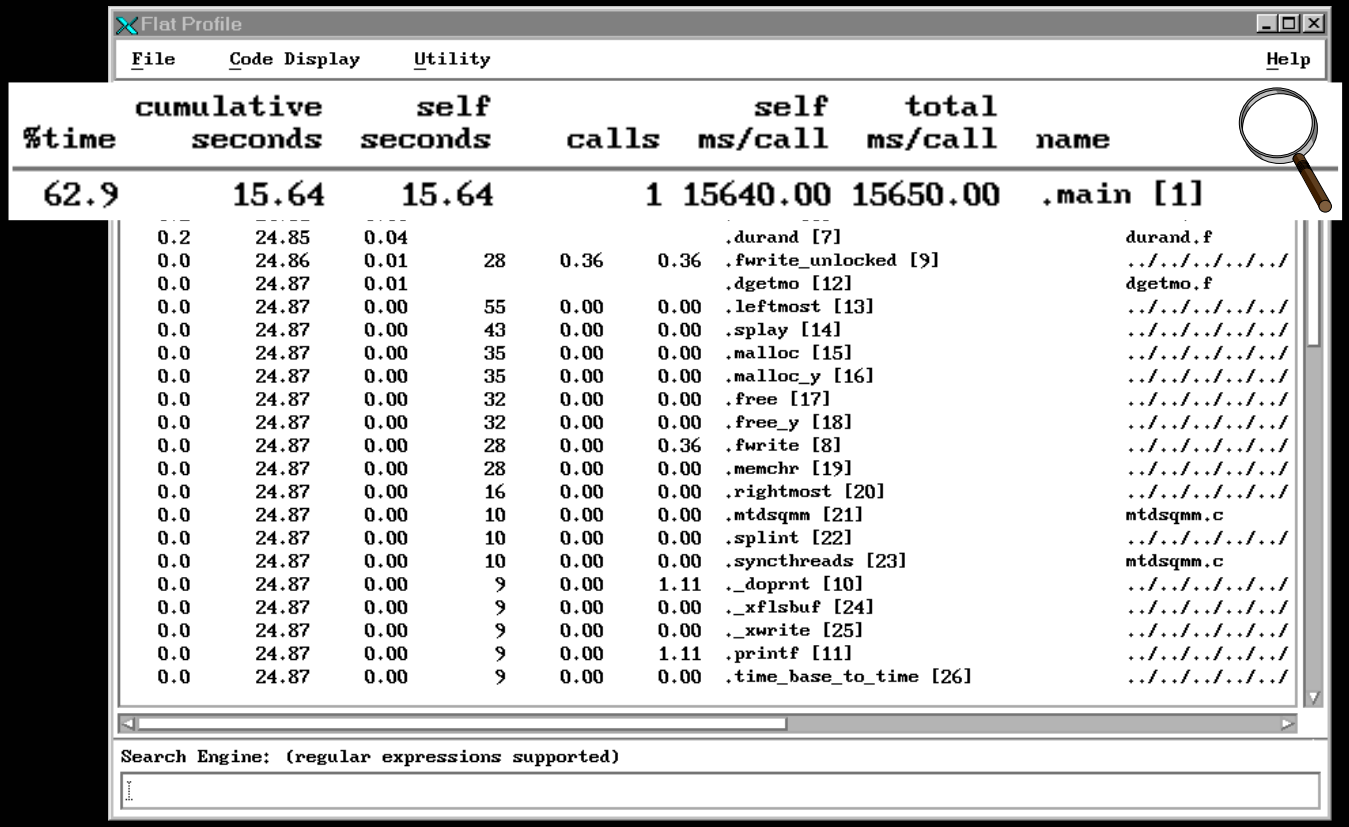


Xprofiler: Zoom In



Xprofiler: Flat Profile

- **Menu `Report` provides usual gprof reports plus some extra ones**
  - Flat Profile
  - Call Graph Profile
  - Function Index
  - Function Call Summary
  - Library Statistics

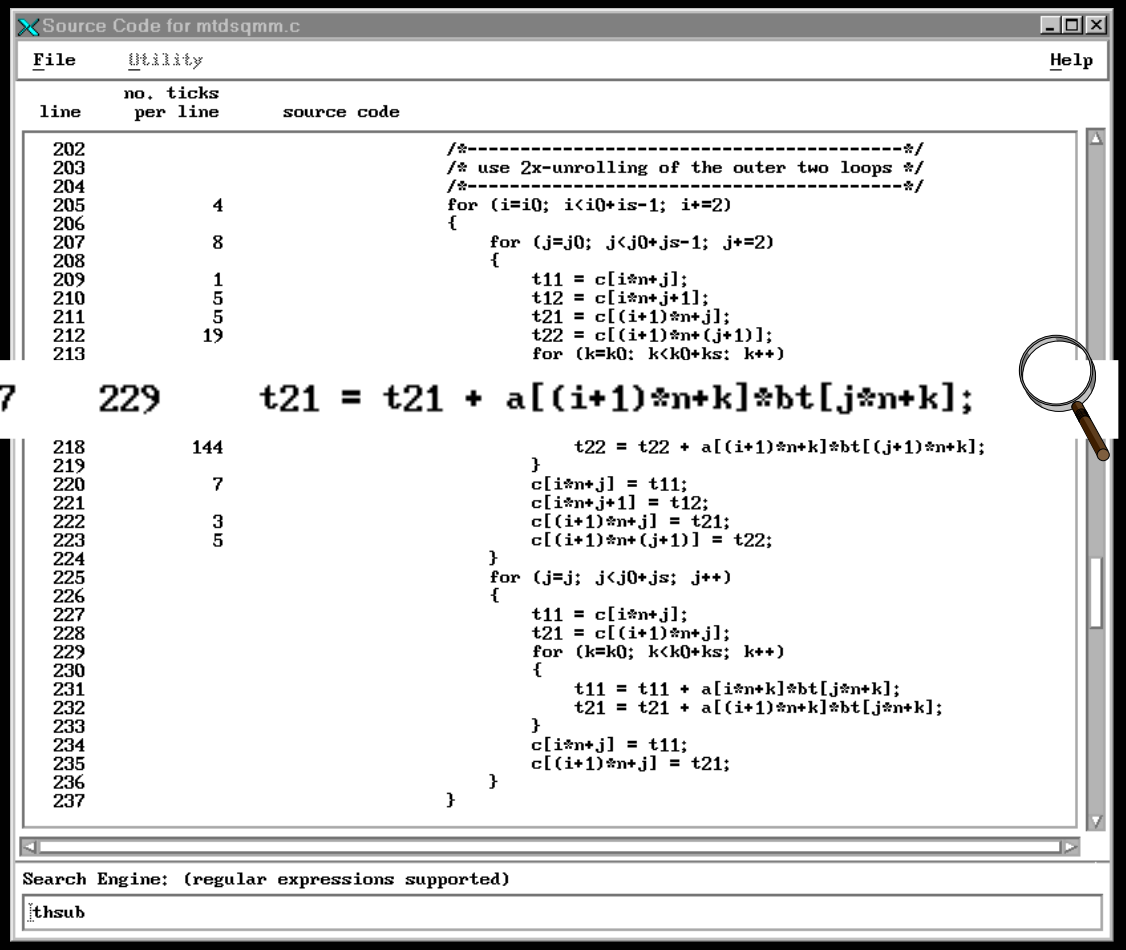


%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name	
62.9	15.64	15.64	1	15640.00	15650.00	.main [1]	
0.2	24.85	0.04				.durand [7]	durand.f
0.0	24.86	0.01	28	0.36	0.36	.fwrite_unlocked [9]	../../../../../
0.0	24.87	0.01				.dgetmo [12]	dgetmo.f
0.0	24.87	0.00	55	0.00	0.00	.leftmost [13]	../../../../../
0.0	24.87	0.00	43	0.00	0.00	.splay [14]	../../../../../
0.0	24.87	0.00	35	0.00	0.00	.malloc [15]	../../../../../
0.0	24.87	0.00	35	0.00	0.00	.malloc_y [16]	../../../../../
0.0	24.87	0.00	32	0.00	0.00	.free [17]	../../../../../
0.0	24.87	0.00	32	0.00	0.00	.free_y [18]	../../../../../
0.0	24.87	0.00	28	0.00	0.36	.fwrite [8]	../../../../../
0.0	24.87	0.00	28	0.00	0.00	.memchr [19]	../../../../../
0.0	24.87	0.00	16	0.00	0.00	.rightmost [20]	../../../../../
0.0	24.87	0.00	10	0.00	0.00	.mtdsqmm [21]	mtdsqmm.c
0.0	24.87	0.00	10	0.00	0.00	.splint [22]	../../../../../
0.0	24.87	0.00	10	0.00	0.00	.syncthreads [23]	mtdsqmm.c
0.0	24.87	0.00	9	0.00	1.11	._doprnt [10]	../../../../../
0.0	24.87	0.00	9	0.00	0.00	._xflsbuf [24]	../../../../../
0.0	24.87	0.00	9	0.00	0.00	._xwrite [25]	../../../../../
0.0	24.87	0.00	9	0.00	1.11	.printf [11]	../../../../../
0.0	24.87	0.00	9	0.00	0.00	.time_base_to_time [26]	../../../../../

Xprofiler: Source Code Window

- **Source code window displays source code with time profile (in ticks=0.01 sec)**

- **Access**
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code



Xprofiler - Disassembler Code

Disassembler Code for .calc3 [3]

File Help

address	no. ticks per instr.	instruction	assembler code		source code
10002E18	81	FCC4287C	fnms	6,4,1,5	
10002E1C	64	CCF70008	lfdu	7,0x8(23)	POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-
10002E20	187	C90C0008	lfd	8,0x8(12)	
10002E24	53	C9750008	lfd	11,0x8(21)	UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-
10002E28	89	FD63582A	fa	11,3,11	
10002E2C	63	FD28387C	fnms	9,8,1,7	POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-
10002E30	4	DD5B0008	stfdu	10,0x8(27)	U(I,J) = UNEW(I,J)
10002E34		C9540008	lfd	10,0x8(20)	VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-
10002E38	113	FCCA302A	fa	6,10,6	
10002E3C	27	C8760008	lfd	3,0x8(22)	POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-
10002E40	87	FD8012FA	fma	12,0,11,2	UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-
10002E44	35	DCB90008	stfdu	5,0x8(25)	V(I,J) = VNEW(I,J)
10002E48	4	FC63482A	fa	3,3,9	POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-
10002E4C	12	CD5A0008	lfdu	10,0x8(26)	UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-
10002E50	62	FCC021BA	fma	6,0,6,4	VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-
10002E54	36	C85B0008	lfd	2,0x8(27)	UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-
10002E58	244	DCEC0008	stfdu	7,0x8(12)	P(I,J) = PNEW(I,J)
10002E5C	28	FD0040FA	fma	8,0,3,8	POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-
10002E60		C8990008	lfd	4,0x8(25)	VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-
10002E64	316	DCD40008	stfdu	6,0x8(20)	
10002E68	29	FC62507C	fnms	3,2,1,10	UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-

Search Engine: (regular expressions supported)

Xprofiler: Tips and Hints

- monenable()/mondisable() to sample certain regions
- Simplest when gmon.out.*, executable, and source code are in one directory
  - Select "**Set File Search Path**" on "**File**" menu to set source directory when source, and executable are not in the same directory
  - Can use **-qfullpath** to encode the path of the source files into the binary
- By default, call tree in main display is "clustered"
  - Menu Filter → Uncluster Functions
  - Menu Filter → Hide All Library Calls
- Libraries must match across systems!
  - on measurement nodes
  - on workstation used for display!
- Must sample realistic problem (sampling rate is 1/100 sec)

HPM – HW counter library

HPM: What Are Performance Counters

- **Extra logic inserted in the processor to count specific events**
- **Updated at every cycle**
- **Strengths:**
  - Non-intrusive
  - Very accurate
  - Low overhead
- **Weakness**
  - Provides only hard counts
  - Specific for each processor
  - Access is not well documented
  - Lack of standard and documentation on what is counted

HPM: Hardware Counters

- **8 counters on PPC970 and Power4, 6 counters on Power 5/5+**
  - Several (100+) events per counter
- **48 UPC counters on Blue Gene/L, 328 events**
- **256 UPC counters on Blue Gene/P, ~1000 events**

- **Events can not be selected independently**
  - PPC970: 41 groups, default: 23
  - Power 4: 64 groups, default: 40
  - Power 5: 140 groups (AIX 5.2), 148 groups (AIX 5.3), default 137
  - Power 5+: 152 groups, default 145
  - Blue Gene/L: 16 groups
  - Blue Gene/P: 4 groups

HPM: Hardware Counters

- Cycles
- Instructions
- Floating point instructions
- Integer instructions
- Load/stores
- Cache misses
- TLB misses
- Branch taken / not taken
- Branch mispredictions

- Useful derived metrics
  - **IPC - instructions per cycle**
  - **Float point rate (Mflip/s)**
  - **Computation intensity**
  - **Instructions per load/store**
  - **Load/stores per cache miss**
  - **Cache hit rate**
  - **Loads per load miss**
  - **Stores per store miss**
  - **Loads per TLB miss**
  - **Branches mispredicted %**

- **Derived metrics allow users to correlate the behavior of the application to one or more of the hardware components**
- **One can define threshold values acceptable for metrics and take actions regarding program optimization when values are below the threshold**

Derived Metrics

- Utilization rate
- Total FP load and store operations
- MIPS
- Instructions per cycle/run cycle/load store
- % Instructions dispatched that completed
- Fixed point operations per Cycle or load/stores
- Branches mispredicted percentage
- number of loads per load miss
- number of stores per store miss
- number of load/stores per L1 miss
- L1 cache hit rate
- number of loads per TLB miss
- number of loads/stores per TLB miss

- Total Loads from L2
- L2 load traffic
- L2 load bandwidth per processor
- Estimated latency from loads from L2
- % loads from L2 per cycle
- Total Loads from local L2
- local L2 load traffic
- local L2 load bandwidth per processor
- Estimated latency from loads from local L2
- % loads from local L2 per cycle
- Total Loads from L3
- L3 load traffic
- L3 load bandwidth per processor
- Estimated latency from loads from L3
- ...

CPU/Memory Performance

- **Instrumentation library**
- **Provides performance information for instrumented program sections**
- **Supports multiple instrumentation sections**
- **Supports MPI, threading and mixed mode**
- **Multiple sections may have the same ID**
- **Run-time performance information collection**

LIBHPM

- **Allows to go in the source code and instrument different sections independently**
- **Supports Fortran, C, and C++**
- **For each instrumented section provides:**
  - Total count & duration (wall clock time)
  - Hardware performance counters information
  - Derived metrics
- **Provides resource usage statistics for the total execution of the instrumented program**
- **Supports:**
  - MPI, OpenMP, & pThreads
  - Multiple instrumentation points
  - Nested instrumentation
  - Multiple calls to an instrumented point

Event Sets

- **4 sets (0-3); ~1000 events**
- **Information for**
  - Time
  - FPU
  - L3 memory
  - Processing Unit
  - Tree network
  - Torus network

Instrumentation section

```
hpmInit( tasked, "my program" );
hpmStart( 1, "outer call" );
do_work();
hpmStart( 2, "computing meaning of life" );
do_more_work();
hpmStop( 2 );
hpmStop( 1 );
hpmTerminate( taskID );
```

Using LIBHPM

- **Declaration:**
- **Use:**
  - #include f_hpm.h

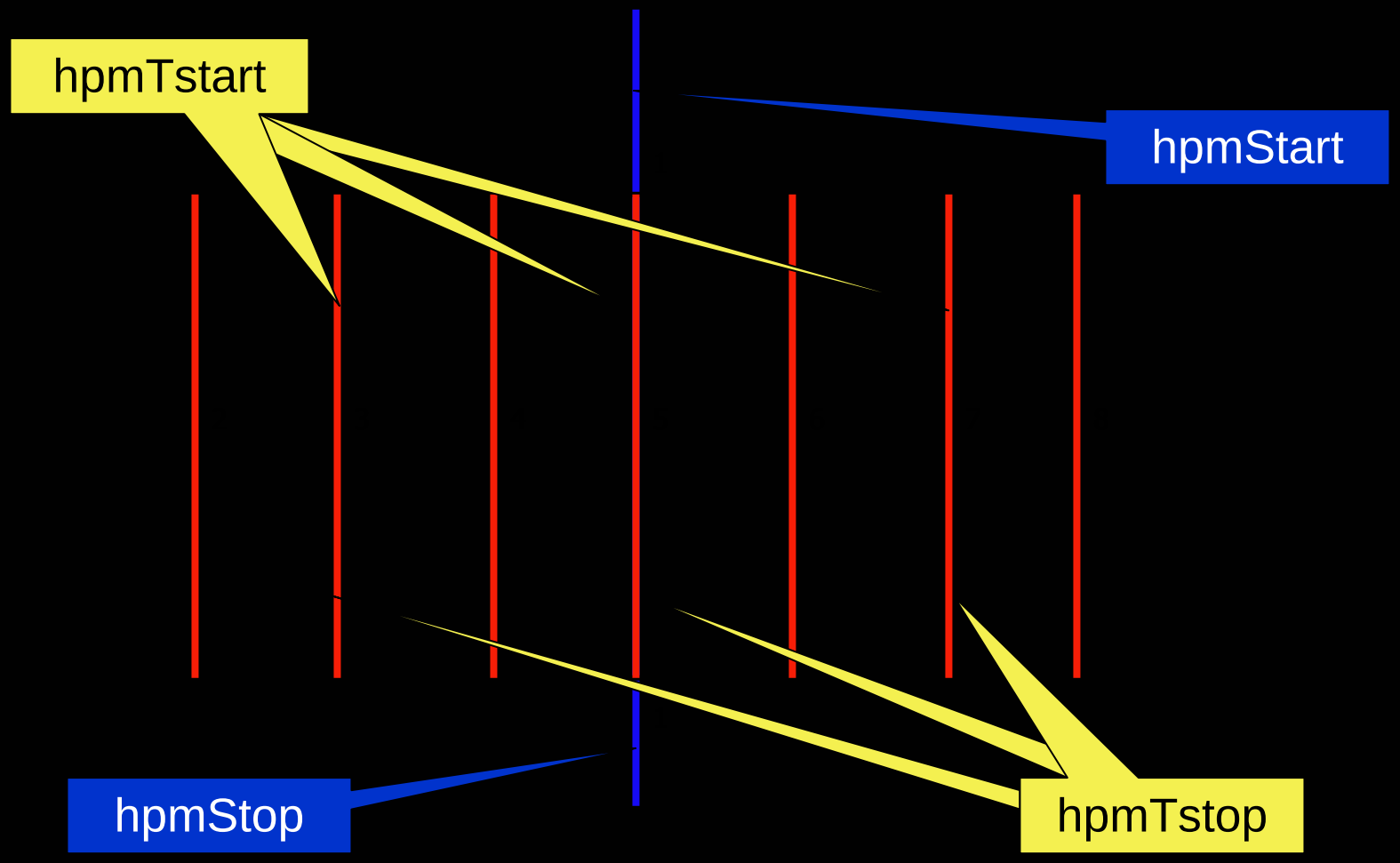
```
call f_hpminit( 0, "prog" )
call f_hpmstart( 1, "work" )
do
  call do_work()
  call f_hpmstart( 22, "more work" )
  call compute_meaning_of_life()
  call f_hpmstop( 22 )
end do
call f_hpmstop( 1 )
call f_hpmterminate( 0 )
```

Use MPI taskID with MPI programs

OpenMP/Threading

- Thread-safe libhpm supports OpenMP and threaded applications.
- A thread-safe linker invocation, such as xlc_r and xlf_r, should be used or
- libpthreads.a must be included in the list of libraries.

HPM: Multi-thread Support



Functions

- **hpmInit(taskID, progName) / f_hpminit(taskID, progName)**
  - taskID is an integer value indicating the node ID.
  - progName is a string with the program name.
- **hpmStart(instID, label) / f_hpmstart(instID, label)**
  - instID is the instrumented section ID. It should be > 0 and <= 100 (can be overridden)
  - Label is a string containing a label, which is displayed by PeekPerf.
- **hpmStop(instID) / f_hpmstop(instID)**
  - For each call to hpmStart, there should be a corresponding call to hpmStop with matching instID
- **hpmTerminate(taskID) / f_hpmterminate(taskID)**
  - This function will generate the output. If the program exits without calling hpmTerminate, no performance information will be generated.

Overhead

- libhpm collects information and performs summarization during run time
  - **there can be considerable overhead if instrumentation sections are inserted inside inner loops.**
- Guideline
  - **If the overhead is several orders of magnitude smaller than the total duration of the instrumented section, you can safely ignore the overhead timing.**
  - **If the overhead is in the same order as the total duration of the instrumented section, you should be suspicious of the results.**
  - **If the overhead is within 20% of the measured wall clock time, a warning is printed to the ASCII output file.**

C and C++ example

declaration:

```
#include "libhpm.h"
```

use:

```
hpmInit( tasked, "my program" );
hpmStart( 1, "outer call" );
do_work();
hpmStart( 2, "computing meaning of life" );
do_more_work();
hpmStop( 2 );
hpmStop( 1 );
hpmTerminate( taskID );
```

Fortran example

```
declaration:
#include "f_hpm.h"
use:
call f_hpminit( taskID, "my program" )
call f_hpmstart( 1, "Do Loop" )
do …
call do_work()
call f_hpmstart( 5, "computing meaning of life" );
call do_more_work();
call f_hpmstop( 5 );
end do
call f_hpmstop( 1 )
call f_hpmterminate( taskID )
```

Multithreaded program

```
!$OMP PARALLEL
!$OMP&PRIVATE (instID)
instID = 30+omp_get_thread_num()
call f_hpmtstart( instID, "computing meaning of life" )
!$OMP DO
do ...
do_work()
end do
call f_hpmtstop( instID )
!$OMP END PARALLEL
```

- If two threads use the same ID numbers for call to **hpmTstart** or **hpmTstop**, libhpm exits with the following error message:
  - **hpmcount ERROR - Instance ID on wrong thread**

Output

- **HPM_OUTPUT_NAME**
- The name <name> is expanded into different file names:
  - ***<name>.hpm* is the file name for ASCII output, which is a one-to-one copy of the screen output.**
  - ***<name>.viz* is the file name for XML output.**
- **HPM_UNIQUE_FILE_NAME**
  - The following string is inserted before the last dot (.) in the file name:
  - _<hostname>_<process_id>_<date>_<time>

Considerations for MPI parallel programs

- HPM_AGGREGATE
  - **does aggregation**
  - **restricting the output to a subset of MPI tasks**
  - takes a value, which is the name of a plug-in that defines the aggregation strategy
- plug-in
  - **a shared object**
  - ***distributor* and *aggregator***

Distributor

- A subroutine that determines the MPI task ID
- Sets or resets environment variables accordingly
- Environment variable can be any environment variable
- The distributor is called before any environment variable is evaluated by HPM

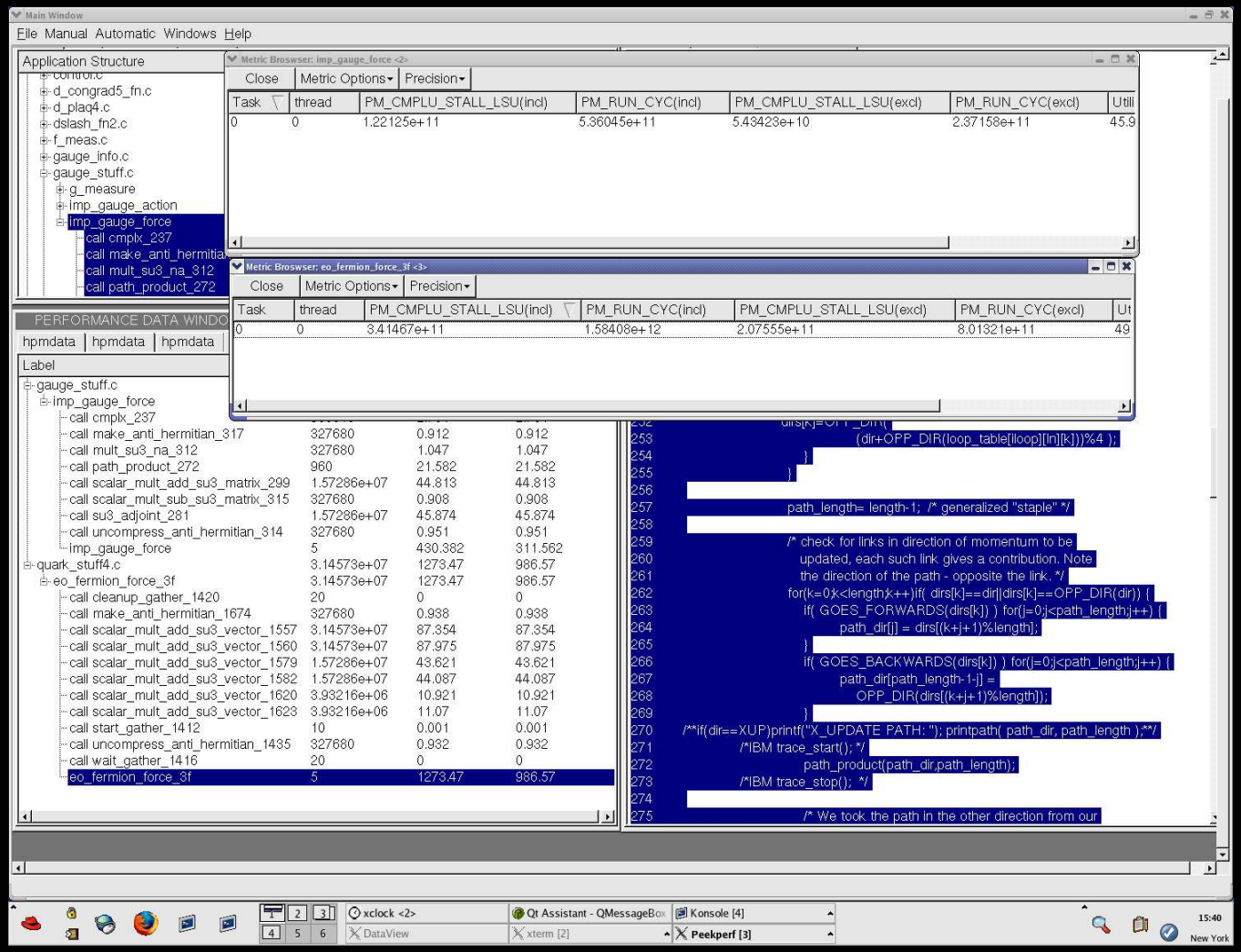
Aggregator

- Aggregation of the hardware counter data across the MPI tasks
- After the hardware counter data is gathered
- Before the data is printed
- Before the derived metrics are computed
- Check Redbook for detailed interface description

Plug-in shipped with HPCT

- *mirror.so*
  - **the plug-in that is called when no plug-in is requested.**
- *loc merge.so*
  - **does a local merge on each MPI task separately**
- *single.so*
  - **does the same as mirror.so, but only on MPI task 0. The output on all other tasks is discarded.**
- *average.so*
  - **a plug-in for taking averages across MPI tasks**

Hardware Counter Performance Visualization



MPI Profiler/Tracer

Message-Passing Performance

MPI Profiler/Tracer

- Implements wrappers around MPI calls using the PMPI interface
  - start timer
  - call pmpi equivalent function
  - stop timer
- Captures MPI calls with source code traceback
- No changes to source code, but MUST compile with -g
- Microsecond order of magnitude overhead per MPI call
- Does not synchronize MPI calls
- Compile with –g and link with libmpitrace.a
- Generate XML files for peekperf

Message-Passing Performance

MPI Tracer

- Captures “timestamped” data for MPI calls with source traceback
- Provides a color-coded trace of execution
- Very useful to identify load-balancing issues

Compiling and Linking

- Consider turning off or having a lower level of optimization (-O2, -O1,...)
  - **High level optimization affects the correctness of the debugging information and can also affect the call stack behavior.**
- To link the application with the library
  - **The option -L/path/to/libraries, where */path/to/libraries* is the path where the libraries are located**
  - **The option -lmpitrace, which should be before the MPI library -lmpich, in the linking order**
  - **The option -llicense to link the license library**

Environment Flags

- **TRACE_ALL_EVENTS (default yes)**
  - saves a record of all MPI events one after MPI Init(), until the application completes or until the trace buffer is full.
  - By default, for MPI ranks 0-255, or for all MPI ranks, if there are 256 or fewer processes in MPI_COMM_WORLD.
  - Alternative: trace_start/stop()
- **MAX_TRACE_EVENTS (Default: 30,000)**
- **TRACE_ALL_TASKS**
  - Set to “yes” to trace all tasks/ranks
- **MAX_TRACE_RANK**
- **TRACEBACK_LEVEL (Default: 0)**
  - Level of trace back the caller in the stack
  - Used to skipped wrappers
- **TRACE_SEND_PATTERN**
  - Has to set to “yes” to trace communication pattern (e.g., Average hops)
  - AverageHops = sum(Hopsi × Bytesi)/sum(Bytesi)

MPI Profiler Output

Main Window

File Manual Automatic Windows Tool

DATA VISUALIZATION WINDOW

mpidata

Label	Count	WallClock	Transfered Bytes
SUMMARY			
MPI_Irecv	1000	0.009671	2.21952e+07
MPI_Isend	1000	0.025689	2.21952e+07
mhd.F			
neighbours(mhd.F)			
MPI_Irecv_631	50	0.000579	614400
MPI_Irecv_636	50	0.000394	614400
MPI_Irecv_669	50	0.000426	614400
MPI_Irecv_674	50	0.000448	614400
MPI_Irecv_716	200	0.001829	4.9152e+06
MPI_Irecv_721	200	0.0019	4.9152e+06
MPI_Irecv_771	200	0.001862	4.9536e+06
MPI_Irecv_776	200	0.002233	4.9536e+06
MPI_Isend_629	50	0.00313	614400
MPI_Isend_634	50	0.000426	614400
MPI_Isend_667	50	0.000753	614400
MPI_Isend_672	50	0.000592	614400
MPI_Isend_714	200	0.005746	4.9152e+06
MPI_Isend_719	200	0.002964	4.9152e+06
MPI_Isend_769	200	0.009673	4.9536e+06
MPI_Isend_774	200	0.002405	4.9536e+06

Metric Broswser: MPI_Irecv_669

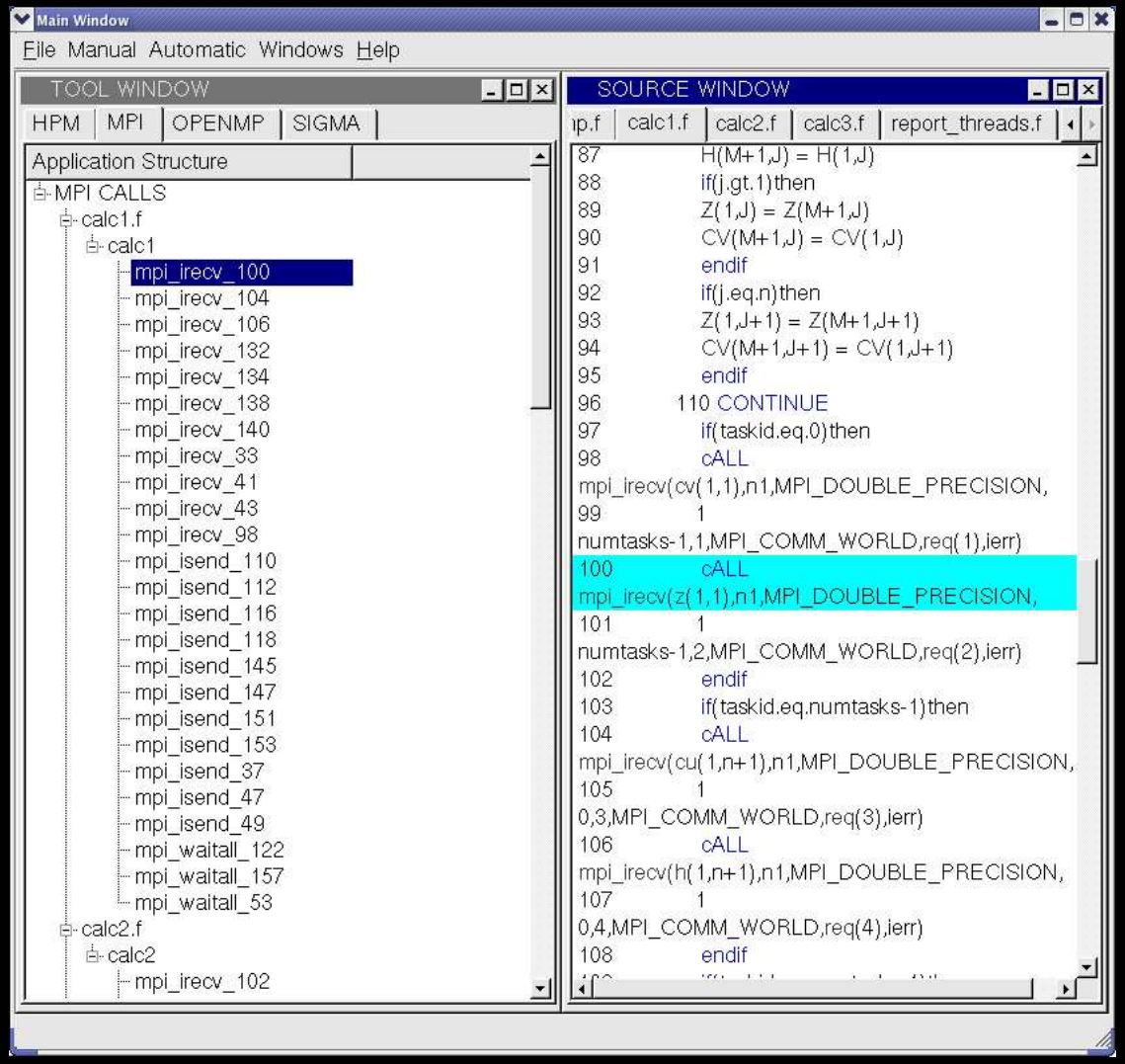
Close | Metric Options | Precision

Task	Count	WallClock	Transfered Bytes
0	50	0.000426	614400
1	50	0.000245	614400
2	50	0.000565	614400
3	50	0.000368	614400

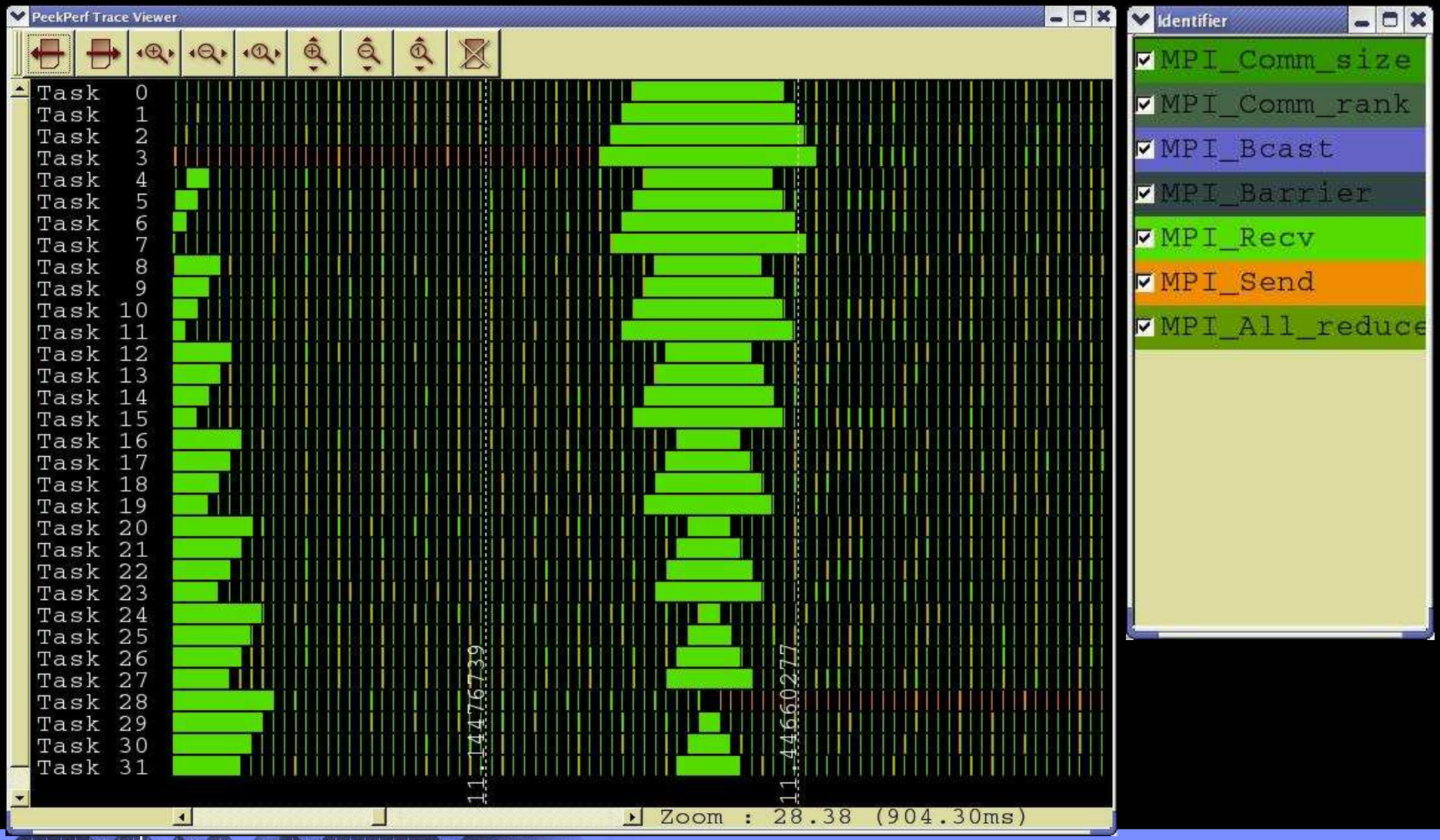
mhd.F

```
655
656
657
658
659
660
661
662
663
664        temp2s1(ilength2+i) = geq(i,jsta,7,2)
665      enddo
666
667      CALL
MPI_ISEND(temp1s2,ilength3,MPI_DOUBLE_PRECISION,jnext
&
668        1        ,comm2d,isend(3),ierr)
669      CALL
MPI_IRECV(temp1r2,ilength3,MPI_DOUBLE_PRECISION,jprev
&
670        1          ,comm2d,irecv(3),ierr)
671
672      CALL
MPI_ISEND(temp2s1,ilength3,MPI_DOUBLE_PRECISION,jprev
&
673        1              ,comm2d,isend(4),ierr)
674      CALL
```

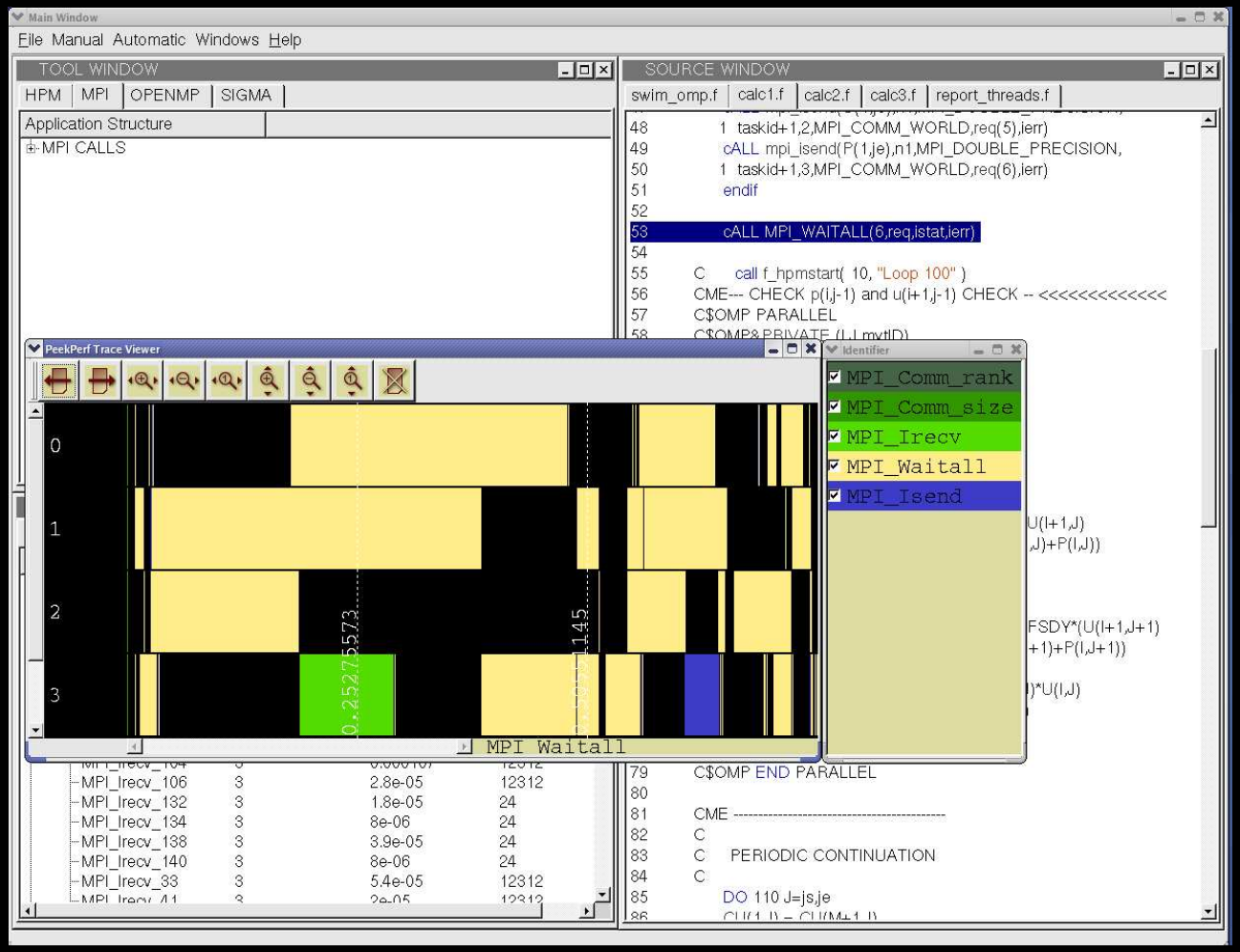
MPI Profile Visualization



MPI Tracer output



MPI Trace Visualization



MPI Message Size Distribution

MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Comm_size	1 (1)	0 ... 4	0	1E-07
MPI_Comm_rank	1 (1)	0 ... 4	0	1E-07
MPI_Isend	2 (1)	0 ... 4	3	0.000006
MPI_Isend	2 (2)	5 ... 16	12	1.4E-06
MPI_Isend	2 (3)	17 ... 64	48	1.3E-06
MPI_Isend	2 (4)	65 ... 256	192	1.3E-06
MPI_Isend	2 (5)	257 ... 1K	768	1.3E-06
MPI_Isend	2 (6)	1K ... 4K	3072	1.3E-06
MPI_Isend	2 (7)	4K ... 16K	12288	1.3E-06
MPI_Isend	2 (8)	16K ... 64K	49152	1.3E-06
MPI_Isend	2 (9)	64K ... 256K	196608	1.7E-06
MPI_Isend	2 (A)	256K ... 1M	786432	1.7E-06
MPI_Isend	1 (B)	1M ... 4M	1048576	9E-07

MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Irecv	2 (1)	0 ... 4	3	4.7E-06
MPI_Irecv	2 (2)	5 ... 16	12	1.4E-06
MPI_Irecv	2 (3)	17 ... 64	48	1.5E-06
MPI_Irecv	2 (4)	65 ... 256	192	2.4E-06
MPI_Irecv	2 (5)	257 ... 1K	768	2.6E-06
MPI_Irecv	2 (6)	1K ... 4K	3072	3.4E-06
MPI_Irecv	2 (7)	4K ... 16K	12288	7.1E-06
MPI_Irecv	2 (8)	16K ... 64K	49152	2.23E-05
MPI_Irecv	2 (9)	64K ... 256K	196608	9.98E-05
MPI_Irecv	2 (A)	256K ... 1M	786432	0.00039
MPI_Irecv	1 (B)	1M ... 4M	1048576	0.000517
MPI_Waitall	21 (1)	0 ... 4	0	1.98E-05
MPI_Barrier	5 (1)	0 ... 4	0	7.8E-06

Communication Summary

$$AverageHops = \frac{\sum_i Hops_i \times Bytes_i}{\sum_i Bytes_i}$$

Communication summary for all tasks:

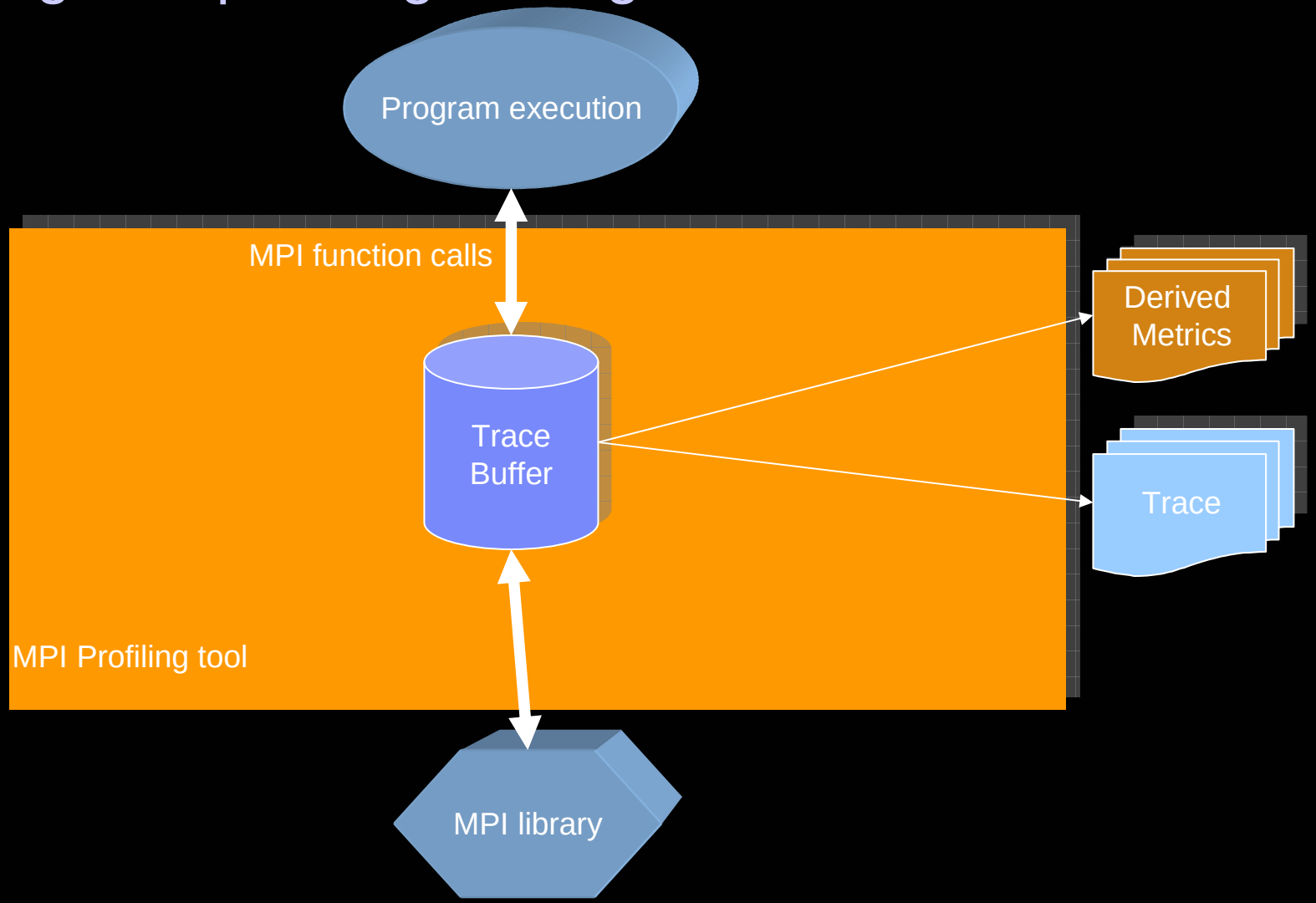
minimum communication time = 0.015 sec for task 0
median communication time = 5.016 sec for task 30
maximum communication time = 5.016 sec for task 20

taskid	xcoord	ycoord	zcoord	procid	total_comm(sec)	avg_hops
0	0	0	0	0	0.015	1
1	1	0	0	0	5.016	1
2	2	0	0	0	5.016	1
3	3	0	0	0	5.016	4
4	0	1	0	0	5.016	1
5	1	1	0	0	5.016	1
6	2	1	0	0	5.016	1
7	3	1	0	0	5.016	4
8	0	2	0	0	5.016	1
9	1	2	0	0	5.016	1
10	2	2	0	0	5.016	1
11	3	2	0	0	5.016	4
12	0	3	0	0	5.016	1
13	1	3	0	0	5.016	1
14	2	3	0	0	5.016	1
15	3	3	0	0	5.016	7
16	0	0	1	0	5.016	1
…	…	…	…	…	…	…

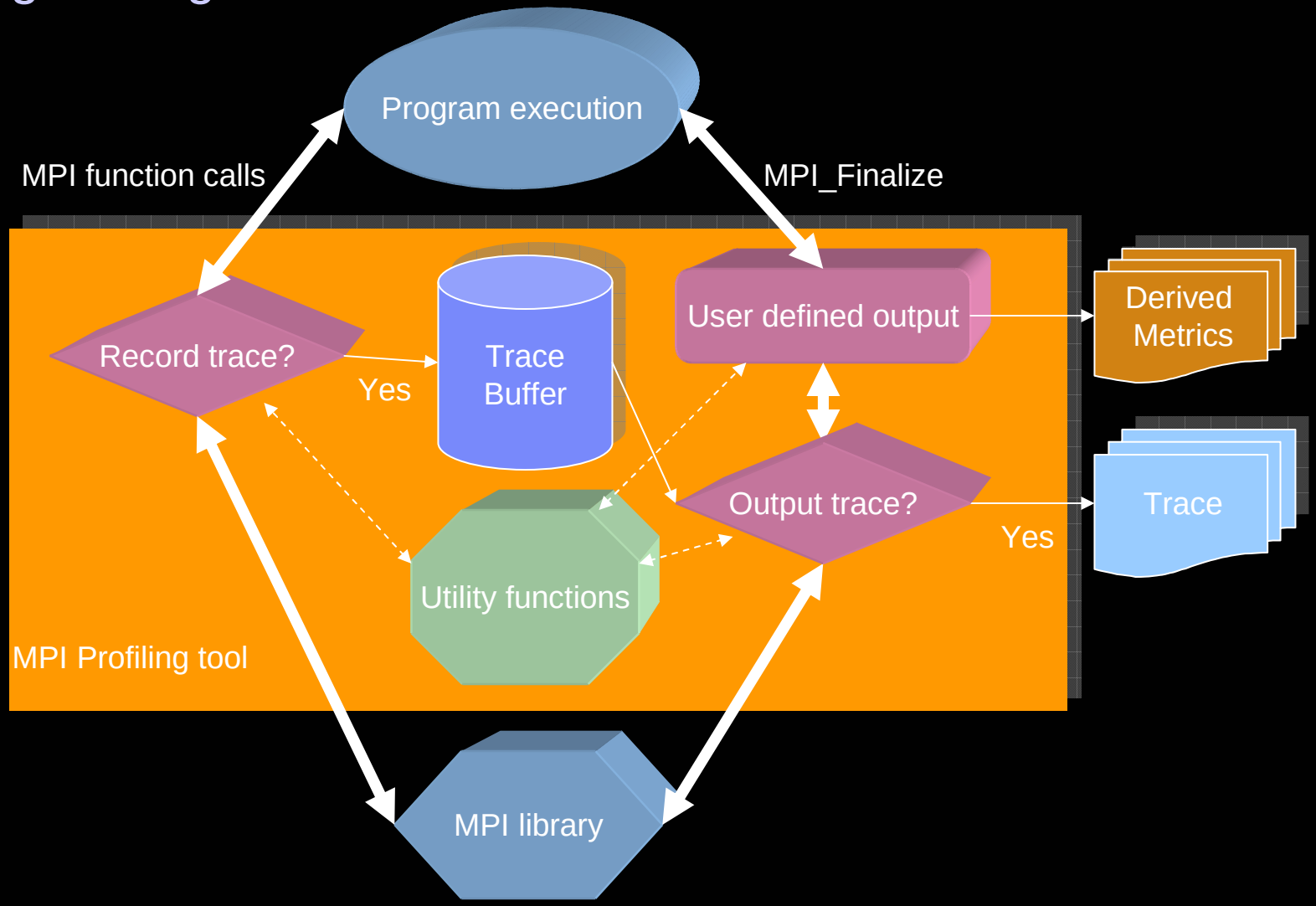
Configuration

- Reduce data volume
  - The volume of trace data can be controlled
  - The cost or overhead to collect, transfer and store will be reduced significantly.
  - Helping to solve the scalability issue
- Provide flexibility
  - Help user focus on interesting points
  - Can be used as a basis towards automatic performance tuning.

Existing MPI profiling/tracing tool



Implementation Example – Programmable MPI profiling/tracing tool

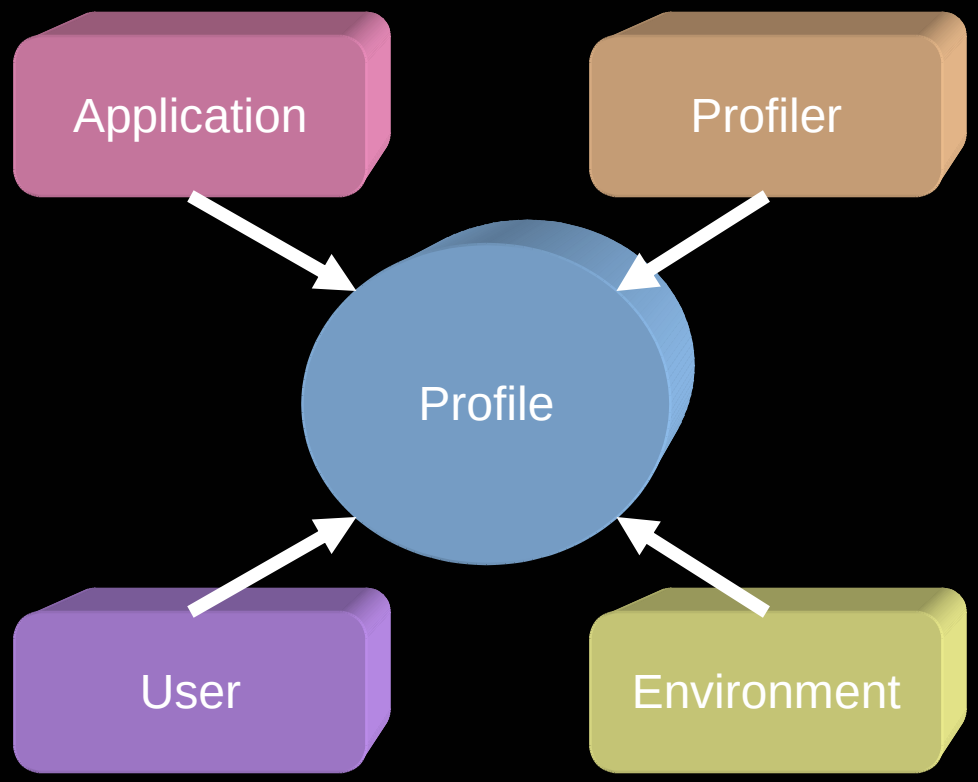


Configuration Functions

- **MT_trace_event()**
  - Called by every profiled MPI functions
  - Decide whether the information should be logged
- **MT_output_trace()**
  - Called in MPI_Finalize()
  - Decide whether the node should output the trace collected
- **MT_output_text()**
  - Called in MPI_Finalize()
  - Used for customize performance data output (e.g., user-defined metrics)

Utility functions

- **Software specific info.**
  - Code segment



- **"Tool factor"**
  - Memory usage

- **User preference**
  - User-defined metrics

- **System info.**
  - Node location

Utility Functions

- **Help user configure the profiling/tracing tool**
- **Information include**
  - MPI functions (call counts, size/distance of data transferred)
  - Time
  - Memory usage
  - Compute node environment (coordinates…)
  - Statistics

Utility Functions

- long long MT_get_mpi_counts(int); /* number of calls for a MPI */
- double MT_get_mpi_bytes(int); /* size of data tranfer for a MPI */
- double MT_get_mpi_time(int); /* time used for a MPI */
- double MT_get_avg_hops(void); /* average hops each MPI message travels */
- double MT_get_time(void); /* time from the MPI_Init */
- double MT_get_elapsed_time(void); /* time between MPI_Finalize and MPI_Init */
- char *MT_get_mpi_name(int); /* name for a MPI */
- int MT_get_tracebufferinfo(struct …); /* info for trace buffer */
- int MT_get_memoryinfo(struct …); /* info for the stack/heap */
- int MT_get_calleraddress(void); /* info for the caller */
- int MT_get_callerinfo(int, struct …); /* get caller detailed info */
- void MT_get_environment(struct …); /* self id info e.g., mpi rank */

Example Usage

```
int MT_trace_event( int id ) {

  /* collect call count distribution */
  for ( i=0; i< env.nmpi; i++ )
    event_count[i] = MT_get_mpi_counts(i);

  if ( compare_distribution( event_count,env.nmpi,0.5 ) == 1 )
    return 0; /* no trace recording if
                 call count distribution stays the same */
  else
    return 1; /* record trace if new call count distribution */
}

int MT_output_trace( int rank ) {
  if ( rank < 32 ) return 1;
          else return 0;
}
```

Modular I/O (MIO)

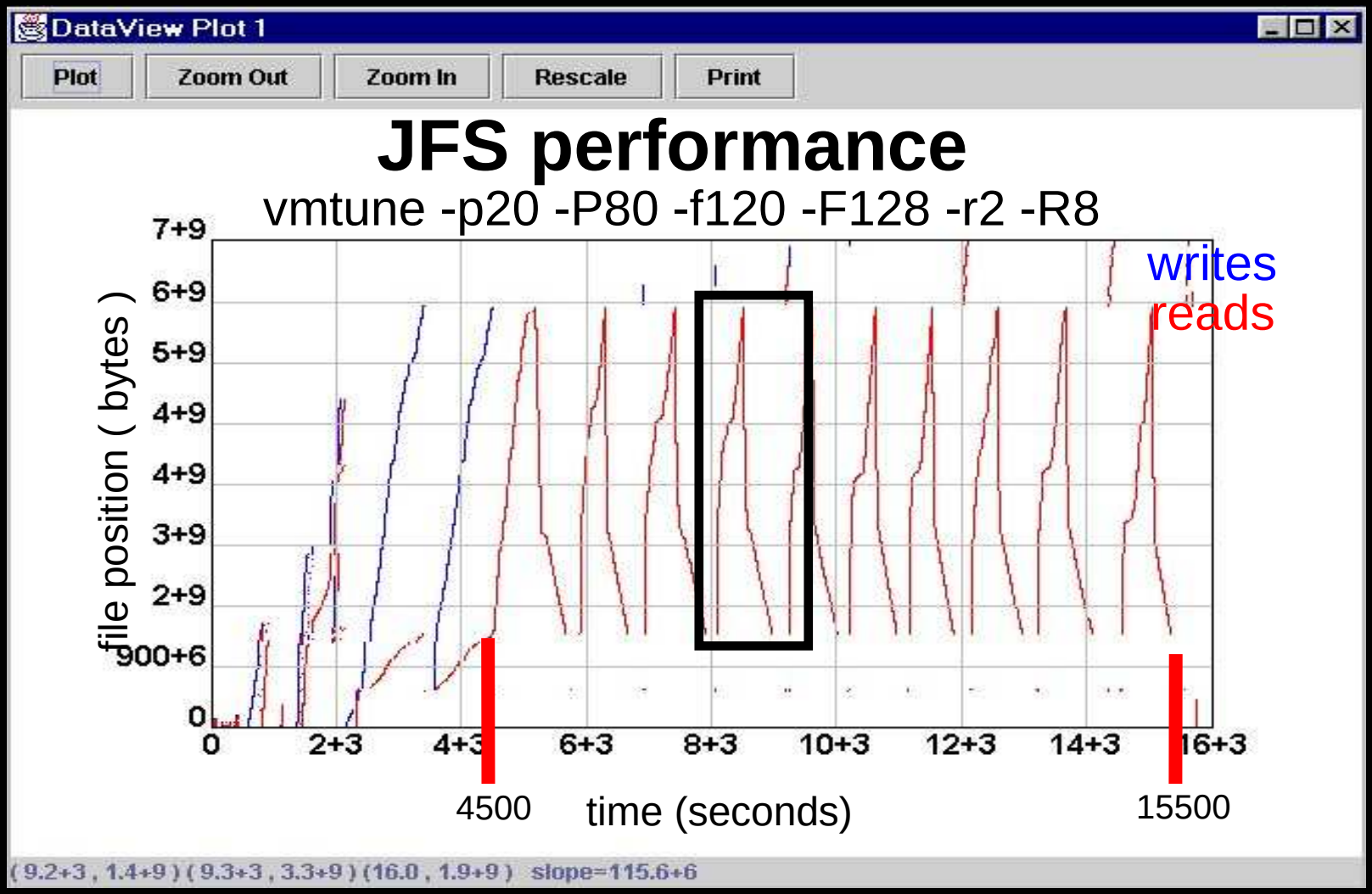
Modular I/O (MIO)

- **Addresses the need of application-level optimization for I/O.**
- **Analyze and tune I/O at the application level**
  - For example, when an application exhibits the I/O pattern of sequential reading of large files
  - MIO
    - Detects the behavior
    - Invokes its asynchronous prefetching module to prefetch user data.
- **Work in progress - Integration into HPC Toolkit with PeekPerf capabilities**
  - Source code traceback
  - Future capability for dynamic I/O instrumentation

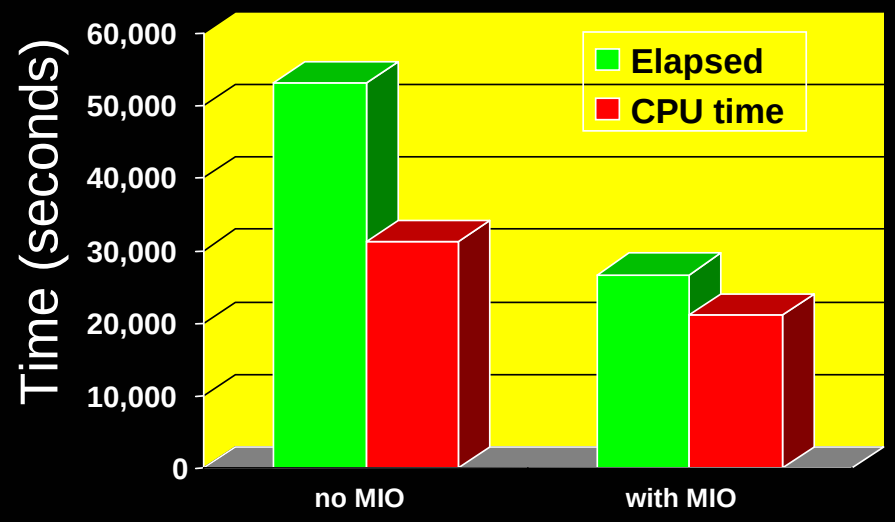
Modular I/O Performance Tool (MIO)

- **I/O Analysis**
  - Trace module
  - Summary of File I/O Activity + Binary Events File
  - Low CPU overhead
- **I/O Performance Enhancement Library**
  - Prefetch module (optimizes asynchronous prefetch and write-behind)
  - System Buffer Bypass capability
  - User controlled pages (size and number)
- **Recoverable Error Handling**
  - Recover module (monitors return values and errnor + reissues failed requests)
- **Remote Data Server**
  - Remote module (simple socket protocol for moving data)

Performance Visualization (work in progress)



MSC.Nastran V2001



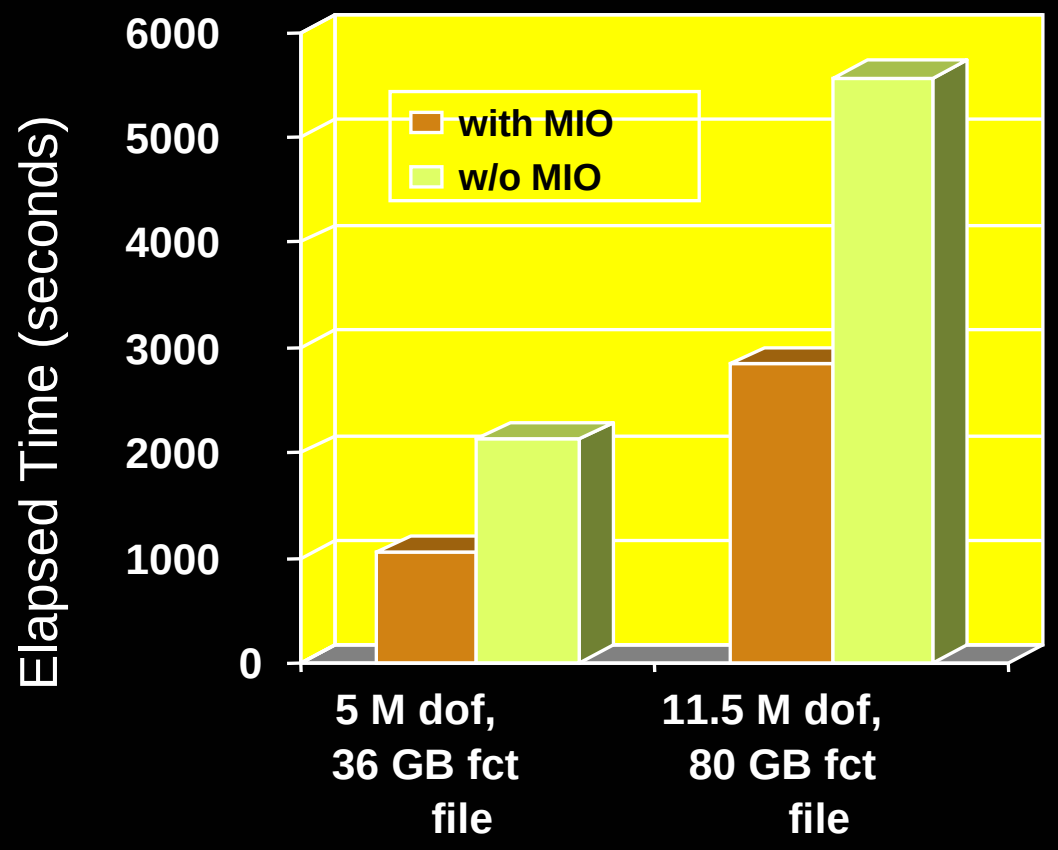
Benchmark:
SOL 111, 1.7M DOF, 1578 modes, 146 frequencies, residual flexibility and acoustics. 120 GB of disk space.

Machine:
4-way, 1.3 GHz p655, 32 GB with 16 GB large pages, JFS striped on 16 SCSI disks.

MSC.Nastran:
V2001.0.9 with large pages, dmp=2 parallel=2 mem=700mb
The run with MIO used mio=1000mb

6.8 TB of I/O in 26666 seconds is an average of about 250 MB/sec

ABAQUS Standard v6.3-4



Engine models

Parallel direct solver

16 POWER4 processors

PompProf

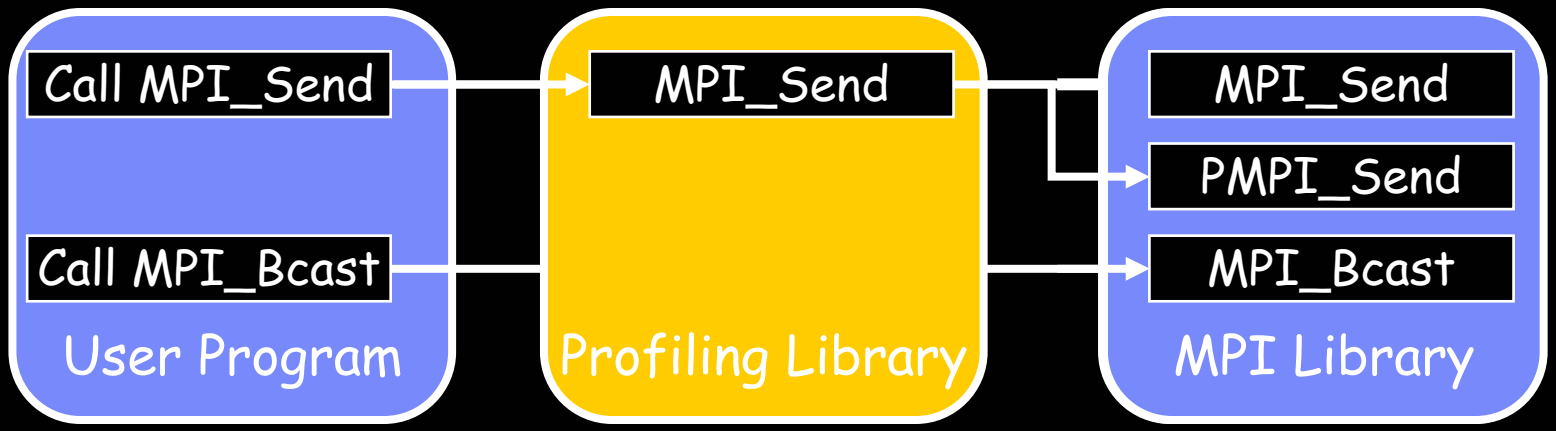
"Standard" OpenMP Monitoring API?

- **Problem:**
  - OpenMP (unlike MPI) does not define standard monitoring interface (at SC06 they accepted a proposal from SUN and others)
  - OpenMP is defined mainly by directives/pragmas
- **Solution:**
  - POMP: OpenMP Monitoring Interface
  - Joint Development
    - Forschungszentrum Jülich
    - University of Oregon
  - Presented at EWOMP'01, LACSI'01 and SC'01
    - "The Journal of Supercomputing", 23, Aug. 2002.

Profiling of OpenMP Applications: POMP

- Portable cross-platform/cross-language API to simplify the design and implementation of OpenMP tools

- POMP was motivated by the MPI profiling interface (PMPI)
  - PMPI allows selective replacement of MPI routines at link time
  - Used by most MPI performance tools (including MPI Profiler/Tracer)



POMP Proposal

- **Three groups of events**
  - OpenMP constructs and directives/pragmas
    - Enter/Exit around each OpenMP construct
      - Begin/End around associated body
    - Special case for parallel loops:
      - ChunkBegin/End, IterBegin/End, or IterEvent instead of Begin/End
    - “Single” events for small constructs like atomic or flush
  - OpenMP API calls
    - Enter/Exit events around omp_set_*_lock() functions
    - “single” events for all API functions
  - User functions and regions
- **Allows application programmers to specify and control amount of instrumentation**

Example: POMP Instrumentation

```
 1:  int main() {
 2:     int id;
***     POMP_Init();
 3:
***     { POMP_handle_t pomp_hd1 = 0;
***       int32 pomp_tid = omp_get_thread_num();
***       POMP_Parallel_enter(&pomp_hd1, pomp_tid, -1, 1,
***           "49*type=pregion*file=demo.c*slines=4,4*elines=8,8**");
 4:     #pragma omp parallel private(id)
 5:     {
***       int32 pomp_tid = omp_get_thread_num();
***       POMP_Parallel_begin(pomp_hd1, pomp_tid);
 6:       id = omp_get_thread_num();
 7:       printf("hello from %d\n", id);
***       POMP_Parallel_end(pomp_hd1, pomp_tid);
 8:     }
***       POMP_Parallel_exit(pomp_hd1, pomp_tid);
***     }
***     POMP_Finalize();
 9:  }
```

SIGMA-POMP: Performance Monitoring Interface for OpenMP based on PSIGMA Instrumentation

- **Approach**
  - A POMP implementation using pSigma's binary instrumentation and rewriting
  - Built on top of pSigma
    - Modifies the binary with performance instrumentation
    - No source code or re-compilation required

POMP Profiler (PompProf)

- **Profiler for OpenMP application implemented on top of SIGMA-POMP**
- **Generates a detailed profile describing overheads and time spent by each thread in three key regions of the parallel application:**
  - Parallel regions
  - OpenMP loops inside a parallel region
  - User defined functions
  -
- **Profile data is presented in the form of an XML file that can be visualized with PeekPerf**

peekperf

File Tools

main.f

Label	Count	Excl. Time	Incl. Time	%Total Overhead	%Imbalance	A
├pregion_324	1	54.4638	128.508	4.1e-05	6.3e-05	12
├loop_1125	10	13.3219	13.3219	0.005431	88.4638	21
├loop_852	10	12.854	12.854	0.005178	52.5563	17
├loop_549	10	12.3907	12.3907	0.005676	98.8048	21
├loop_1685	10	12.3036	12.3036	0.005682	62.1319	16
├loop_1408	10	11.8975	11.8975	0.005578	96.0548	19
├loop_1934	10	10.1843	10.1843	0.006926	41.4999	13
├loop_790	1	0.522151	0.522151	0.013839	37.576	0.
├loop_1668	1	0.485633	0.485633	0.00896	36.8549	0.
├loop_1623	1	0.039318	0.039318	0.596515	2.79627	0.
├loop_1379	1	0.031769	0.031769	0.198128	13.846	0.
├func_rdparam_174	1	0.01605	0.01605	0	0	0
├loop_855	1	0.013306	0.013306	0.224878	342.237	0.
├func_main_155	1	0.000771	128.525	0	0	0
├func_deltat_550	1	2.2e-05	2.2e-05	0	0	0
├func_layout_296	1	6e-06	6e-06	0	0	0
└func_changedir_184	1	3e-06	3e-06	0	0	0

main.f runhyd3.f

```
 &        msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
  endif


c$omp do schedule(static)
      do ip=1,nchunk



    if ((ithread.eq.1).and.(ip.ge.icomm_point(1,4)).and.
 &        (irec1br.eq.0) .and.        (iflag.eq.0)) then
      irec1br = 1

      call bdrys1br(dddo, nz, nx, ny, 5,
 &        msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
 &        msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
 &        msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
      call bdry2o1s(dddo, nz, nx, ny, 5,
 &        msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
                                    _mnybdy,
                                    _mxybdy)

                                    _xp, msg_ym, msg_yp,
                                    _mnybdy,
                                    _mxybdy)


                                    (2,4)).and.
                                    0)) then


                                    _xp, msg_ym, msg_yp,
 &        msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
 &        msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
      call bdry3o1r(dddo, nz, nx, ny, 5,
```

Metric Broswser: loop_1125

Close | Metric Options | Precision

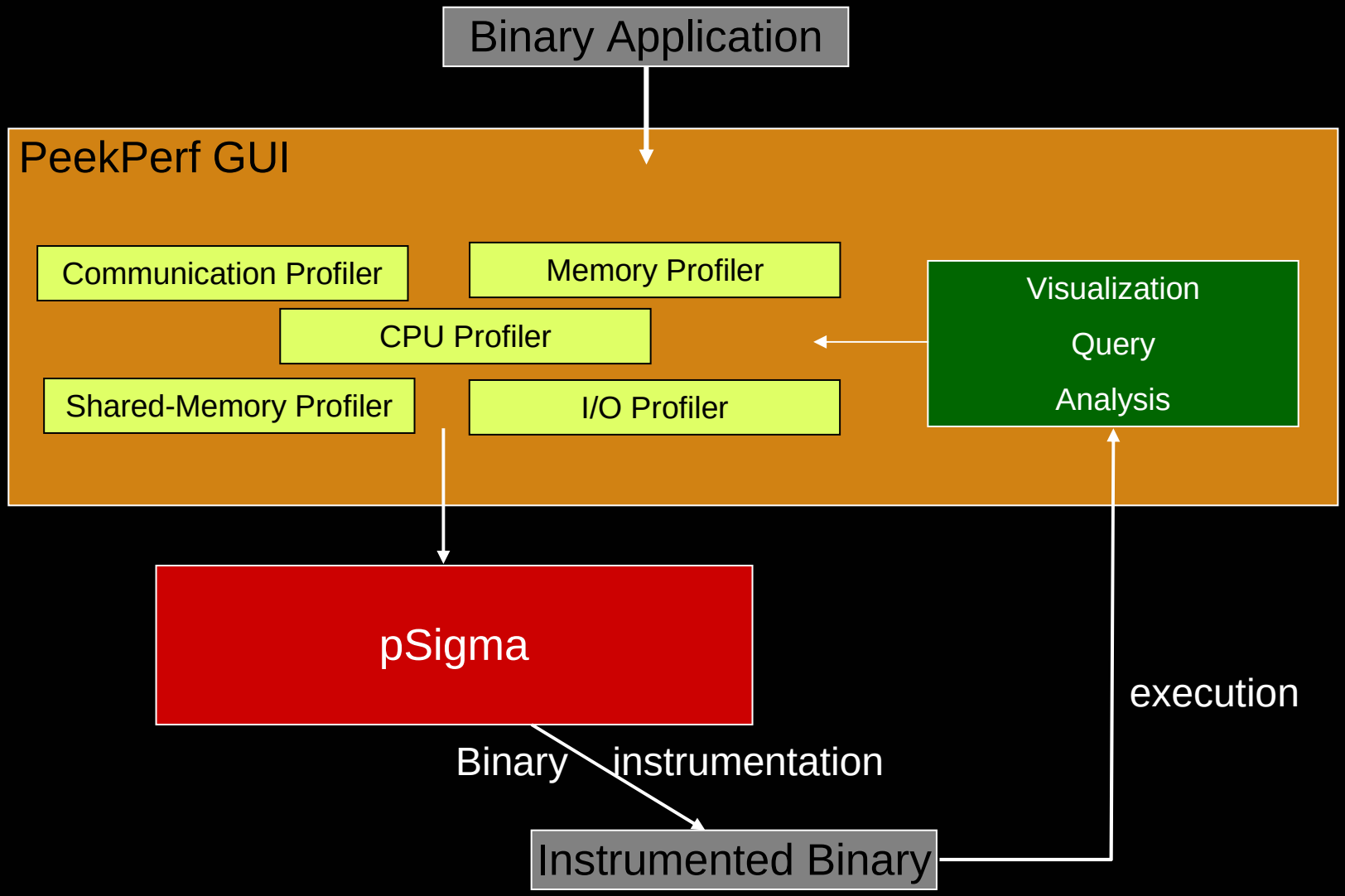
Task	thread	Time in Master	TT: Thread Time	CT: Computation Time	%Imbalance	TO = TT - CT	%TO (Barrier)	%TO (RTL)
0	0	13.3219	13.3219	13.3211	0	0.000723	0.000275	0.005156
0	1	0	15.8615	15.861	19.0664	0.000504	0.000108	0.003679
0	2	0	21.0295	21.029	57.8616	0.000514	0.000103	0.003753
0	3	0	24.4342	24.4338	83.4208	0.000416	4e-06	0.003119
0	4	0	24.1806	24.1802	81.5175	0.000423	0	0.003173
0	5	0	25.0957	25.0952	88.3864	0.00047	1.4e-05	0.00351
0	6	0	25.1062	25.1055	88.4638	0.00062	0.000157	0.004495
0	7	0	23.9859	23.9854	80.0548	0.000556	0.000112	0.004063

Interactive Performance Debugger

Interactive Performance Debugger

- Control instrumentation from the visualization GUI: one complete framework for performance analysis
- Operate on the source code but perform modifications on the binary
- Debugger-like interface
- Automatically display collected data
- Refine instrumentation (iterative tuning)
- Comparison between data and between multiple runs
- Graphics capabilities (tables, charts)
- Query language for “what-if” analysis

Structure of the HPC toolkit



Peekperf Main Interface

Main Window <2>

File Manual Automatic Windows Tool

DATA COLLECTION WINDOW

HPM | MPI | OPENMP | MEMORY | MIO

Application Structure

- MPI CALLS
  - statusesf.f
    - MAIN__
      - mpi_comm_rank_25
      - mpi_finalize_55
      - mpi_init_18
      - mpi_irecv_40
      - mpi_isend_33
      - mpi_waitall_48

DATA VISUALIZATION WINDOW

mpidata

Label	Count	Transfered Bytes	WallClock
statusesf.f			
MAIN__(statusesf.f)			
MPI_Comm_rank_25	1	0	4e-06
MPI_Irecv_40	50	0	8.1e-05
MPI_Isend_33	50	0	0.000184
MPI_Waitall_48	1	0	5.2e-05
SUMMARY			
MPI_Comm_rank	1	0	4e-06
MPI_Irecv	50	0	8.1e-05
MPI_Isend	50	0	0.000184
MPI_Waitall	1	0	5.2e-05

SOURCE CODE WINDOW

statusesf.f

```
9          integer nreqs
10         parameter (nreqs = 100)
11         integer reqs(nreqs)
12         integer ierr, rank, i
13         integer errs
14
15
16         ierr = -1
17         errs = 0
18         call mpi_init( ierr )
19         if (ierr .ne. MPI_SUCCESS) then
20           errs = errs + 1
21           print *, 'Unexpected return from MPI_INIT', ierr
22         endif
23
24         ierr = -1
25         call mpi_comm_rank( MPI_COMM_WORLD, rank, ierr )
26         if (ierr .ne. MPI_SUCCESS) then
27           errs = errs + 1
28           print *, 'Unexpected return from MPI_COMM_WORLD', ierr
29         endif
30         do i=1, nreqs, 2
31           ierr = -1
32           call mpi_isend( MPI_BOTTOM, 0, MPI_BYTE, rank, i,
33       $       MPI_COMM_WORLD, reqs(i), ierr )
34           if (ierr .ne. MPI_SUCCESS) then
35             errs = errs + 1
36             print *, 'Unexpected return from MPI_ISEND', ierr
37           endif
38           ierr = -1
39           call mpi_irecv( MPI_BOTTOM, 0, MPI_BYTE, rank, i,
40       $       MPI_COMM_WORLD, reqs(i+1), ierr )
41           if (ierr .ne. MPI_SUCCESS) then
42             errs = errs + 1
43             print *, 'Unexpected return from MPI_IRECV', ierr
44           endif
45         enddo
```

Peekperf Main Interface (Cont.)

Main Window <2>

File Manual Automatic Windows Tool

DATA COLLECTION WINDOW

HPM | MPI | OPENMP | MEMORY | MIO

Application Structure

- MPI CALLS
  - statusesf.f
    - MAIN__
      - mpi_comm_rank_25
      - mpi_finalize_55
      - mpi_init_18
      - mpi_irecv_40
      - mpi_isend_33
      - mpi_waitall_48

DATA VISUALIZATION WINDOW

mpidata

Label	Count	Transfered Bytes	WallClock
statusesf.f			
MAIN__(statusesf.f)			
MPI_Comm_rank_25	1	0	
MPI_Irecv_40	50	0	
MPI_Isend_33	50	0	
MPI_Waitall_48	1	0	
SUMMARY			
MPI_Comm_rank	1	0	
MPI_Irecv	50	0	
MPI_Isend	50	0	
MPI_Waitall	1	0	

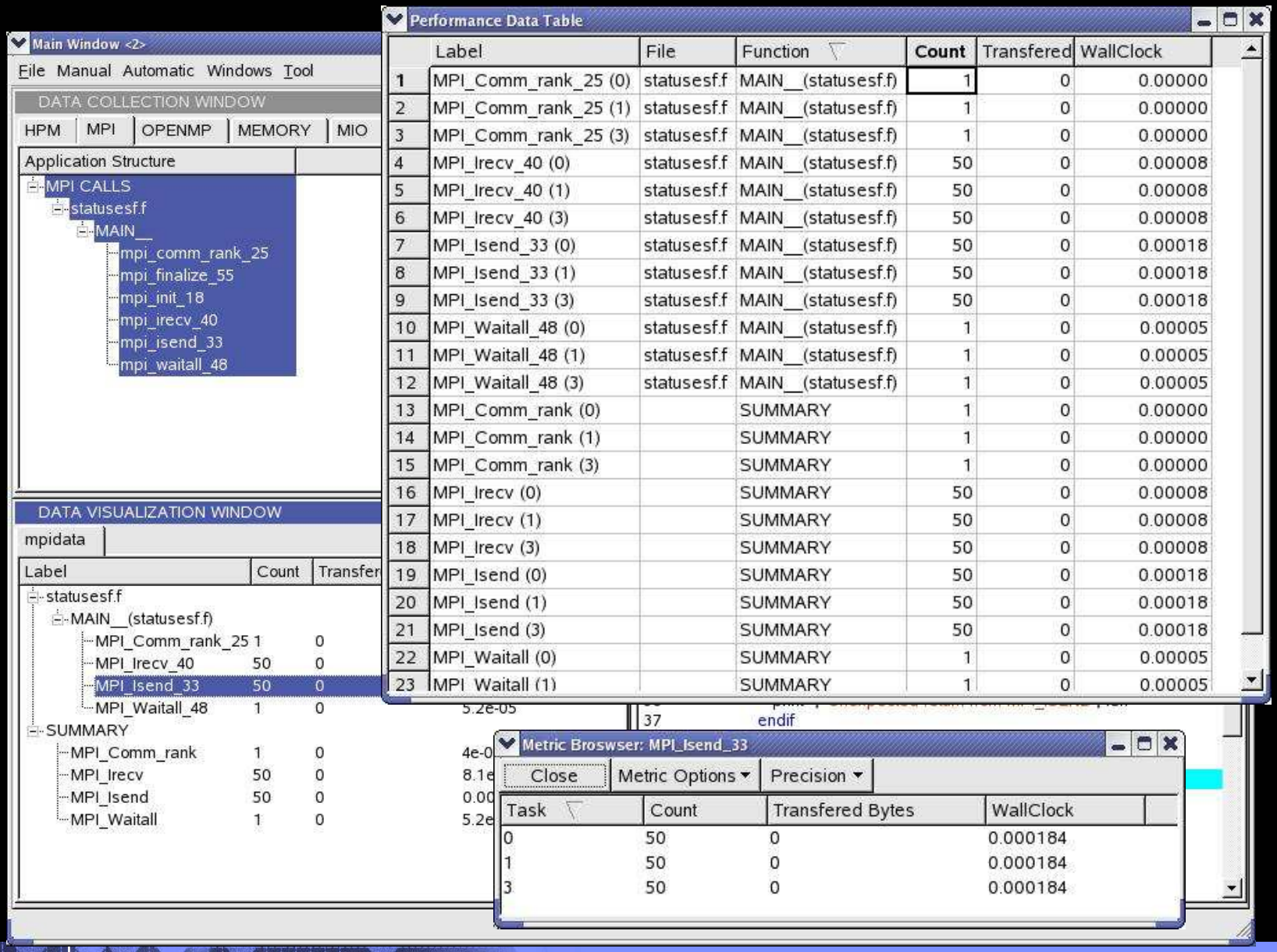
Set Filter
No Filter
Expand the Tree
Collapse the Tree
Show as Table
Show Other Rank
View Tracer

SOURCE CODE WINDOW

statusesf.f

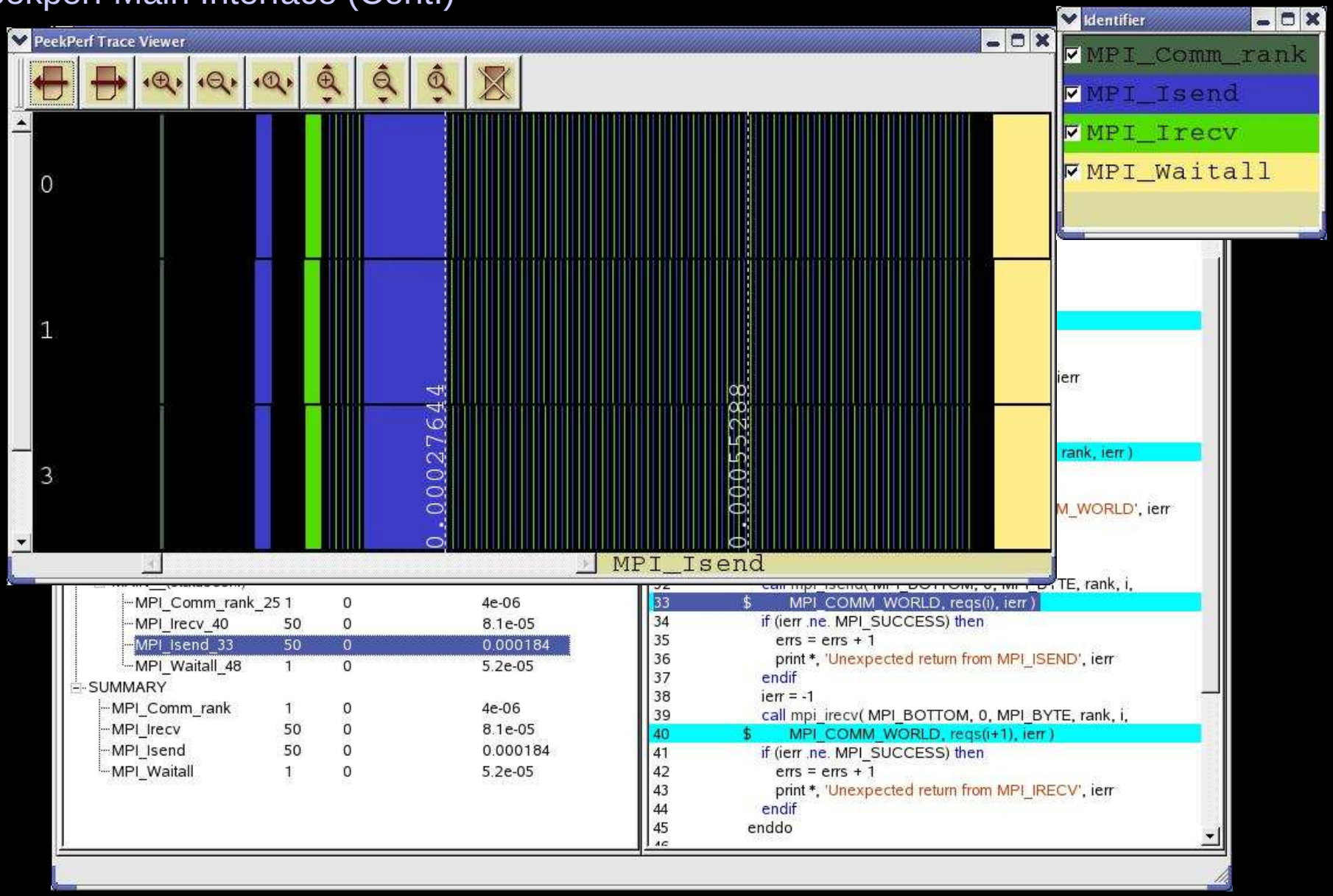
```
9       integer nreqs
10      parameter (nreqs = 100)
11      integer reqs(nreqs)
12      integer ierr, rank, i
13      integer errs
14
15
16      ierr = -1
17      errs = 0
18      call mpi_init( ierr )
19      if (ierr .ne. MPI_SUCCESS) then
20        errs = errs + 1
21        print *, 'Unexpected return from MPI_INIT', ierr
22      endif
23
24      ierr = -1
25      call mpi_comm_rank( MPI_COMM_WORLD, rank, ierr )
26      if (ierr .ne. MPI_SUCCESS) then
27        errs = errs + 1
28        print *, 'Unexpected return from MPI_COMM_WORLD', ierr
29      endif
30      do i=1, nreqs, 2
31        ierr = -1
32        call mpi_isend( MPI_BOTTOM, 0, MPI_BYTE, rank, i,
33     $      MPI_COMM_WORLD, reqs(i), ierr )
34        if (ierr .ne. MPI_SUCCESS) then
35          errs = errs + 1
36          print *, 'Unexpected return from MPI_ISEND', ierr
37        endif
38        ierr = -1
39        call mpi_irecv( MPI_BOTTOM, 0, MPI_BYTE, rank, i,
40     $      MPI_COMM_WORLD, reqs(i+1), ierr )
41        if (ierr .ne. MPI_SUCCESS) then
42          errs = errs + 1
43          print *, 'Unexpected return from MPI_IRECV', ierr
44        endif
45      enddo
```

Peekperf Main Interface (Cont.)



	Label	File	Function	Count	Transfered	WallClock
1	MPI_Comm_rank_25 (0)	statusesf.f	MAIN__(statusesf.f)	1	0	0.00000
2	MPI_Comm_rank_25 (1)	statusesf.f	MAIN__(statusesf.f)	1	0	0.00000
3	MPI_Comm_rank_25 (3)	statusesf.f	MAIN__(statusesf.f)	1	0	0.00000
4	MPI_Irecv_40 (0)	statusesf.f	MAIN__(statusesf.f)	50	0	0.00008
5	MPI_Irecv_40 (1)	statusesf.f	MAIN__(statusesf.f)	50	0	0.00008
6	MPI_Irecv_40 (3)	statusesf.f	MAIN__(statusesf.f)	50	0	0.00008
7	MPI_Isend_33 (0)	statusesf.f	MAIN__(statusesf.f)	50	0	0.00018
8	MPI_Isend_33 (1)	statusesf.f	MAIN__(statusesf.f)	50	0	0.00018
9	MPI_Isend_33 (3)	statusesf.f	MAIN__(statusesf.f)	50	0	0.00018
10	MPI_Waitall_48 (0)	statusesf.f	MAIN__(statusesf.f)	1	0	0.00005
11	MPI_Waitall_48 (1)	statusesf.f	MAIN__(statusesf.f)	1	0	0.00005
12	MPI_Waitall_48 (3)	statusesf.f	MAIN__(statusesf.f)	1	0	0.00005
13	MPI_Comm_rank (0)		SUMMARY	1	0	0.00000
14	MPI_Comm_rank (1)		SUMMARY	1	0	0.00000
15	MPI_Comm_rank (3)		SUMMARY	1	0	0.00000
16	MPI_Irecv (0)		SUMMARY	50	0	0.00008
17	MPI_Irecv (1)		SUMMARY	50	0	0.00008
18	MPI_Irecv (3)		SUMMARY	50	0	0.00008
19	MPI_Isend (0)		SUMMARY	50	0	0.00018
20	MPI_Isend (1)		SUMMARY	50	0	0.00018
21	MPI_Isend (3)		SUMMARY	50	0	0.00018
22	MPI_Waitall (0)		SUMMARY	1	0	0.00005
23	MPI_Waitall (1)		SUMMARY	1	0	0.00005

Peekperf Main Interface (Cont.)



Summery

- The IBM HPC Toolkit provides an integrated framework for performance analysis
- Support iterative analysis and automation of the performance tuning process
- The standardized software layers make it easy to plug in new performance analysis tools
- Operates on the binary and yet provide reports in terms of source-level symbols
- Provides multiple layers that the user can exploit (from low-level instrumentations to high-level performance analysis)
- Full source code traceback capability
- Dynamically activate/deactivate data collection and change what information to collect